sql密码：980107

day01

1.基础

数据库：英文单词DataBase，简称DB

数据库管理系统：DataBaseManagement，简称DBMS。

SQL：结构化查询语言

DBMS--执行--> SQL --操作--> DB

2.安装（端口号）

b站视频

端口号port是任何一个软件/应用都会有的，端口号是应用的唯一代表。

端口号通常和IP地址在一块，**IP地址用来定位计算机的，端口号port是用来定位计算机上某个服务的/某个应用的！**

在同一台计算机上，端口号不能重复。具有唯一性。

mysql数据库启动的时候，这个服务占有的默认端口号是3306这是大家都知道的事儿。记住。

3.卸载

b站视频

4.查看mysql服务

计算机-->右键-->管理-->服务和应用程序-->服务-->找mysql服务
MySQL的服务，默认是“启动”的状态，只有启动了mysql才能用。
默认情况下是“自动”启动，自动启动表示下一次重启操作系统的时候自动启动该服务。

可以在服务上点击右键：
    启动
    重启服务
    停止服务
    ...

还可以改变服务的默认配置：
    服务上点击右键，属性，然后可以选择启动方式：
        自动（延迟启动）
        自动
        手动
        禁用

5.启动和关闭mysql服务

```
net stop 服务名称;
net start 服务名称;
```

6.使用客户端登录mysql数据库

```
C:\Users\Administrator>mysql -uroot -p123456
C:\Users\Administrator>mysql -uroot -p
Enter password: ******
```

7.mysql常用命令

退出mysql ： exit

查看mysql中有哪些数据库？

```
show databases;
```

注意：以分号结尾，分号是英文的分号。

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

mysql默认自带了4个数据库。

怎么选择使用某个数据库呢？

```
mysql> use test;
Database changed
```

表示正在使用一个名字叫做test的数据库。

怎么创建数据库呢？

```
mysql> create database bjpowernode;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bjpowernode        |
| mysql              |
| performance_schema |
```

```
    | test               |
    +--------------------+
```

查看某个数据库下有哪些表？

```
    mysql> show tables;
```

注意：以上的命令不区分大小写，都行。

查看mysql数据库的版本号：

```
mysql> select version();
    +-----------+
    | version() |
    +-----------+
    | 5.5.36    |
    +-----------+
```

查看当前使用的是哪个数据库？

```
mysql> select database();
+-------------+
| database()  |
+-------------+
| bjpowernode |
+-------------+

mysql> show
-> databases
-> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| bjpowernode        |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

```
注意：mysql是不见“;”不执行，“;”表示结束！

mysql> show
->
->
->
->
->
->
->
->
-> \c
mysql>
\c用来终止一条命令的输入。
```

8.表

数据库当中最基本的单元是表：table

什么是表table？为什么用表来存储数据呢？

```
姓名        性别        年龄(列：字段)
---------------------------
张三        男              20              -------->行（记录）
李四        女              21              -------->行（记录）
王五        男              22              -------->行（记录）
```

数据库当中是以表格的形式表示数据的。
因为表比较直观。

任何一张表都有行和列：
行（row）：被称为数据/记录。
列（column）：被称为字段。

姓名字段、性别字段、年龄字段。

了解一下：

每一个字段都有：字段名、数据类型、约束等属性。

字段名可以理解，是一个普通的名字，见名知意就行。

数据类型：字符串，数字，日期等，后期讲。

约束：约束也有很多，其中一个叫做唯一性约束，

这种约束添加之后，该字段中的数据不能重复。

9.SQL语句的分类

SQL语句有很多，最好进行分门别类，这样更容易记忆。

分为：

DQL：

数据查询语言（凡是带有**select关键字**的都是查询语句）

select...

DML：

数据操作语言（凡是对表当中的数据进行增删改的都是DML）

insert delete update

insert 增

delete 删

update 改

这个主要是操作**表中的数据data**。

DDL：

数据定义语言

凡是带有create、drop、alter的都是DDL。

DDL主要操作的是**表的结构。不是表中的数据。**

create：新建，等同于增

drop：删除

alter：修改

这个增删改和DML不同，这个主要是对表结构进行操作。

TCL：

不是王牌电视。

是事务控制语言

包括：

事务提交：commit;

事务回滚：rollback;

DCL：

是数据控制语言。

例如：授权grant、撤销权限revoke....

10.导入数据

```
mysql> source D:\course\03-
MySQL\document\bjpowernode.sql
```

11.关于导入的表bjpowernode

```
mysql> show tables;
+----------------------+
| Tables_in_bjpowernode |
+----------------------+
| dept                 |
| emp                  |
| salgrade             |
+----------------------+

dept是部门表
emp是员工表
salgrade 是工资等级表
```

怎么查看表中的数据呢？

```
    select * from 表名; //统一执行这个SQL语句。

mysql> select * from emp; // 从emp表查询所有数据。
+-------+--------+-----------+------+------------+---
-------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   |
SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---
-------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |
 800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 |
1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 |
1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 |
2975.00 |    NULL |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 |
1250.00 | 1400.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 |
2850.00 |    NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 |
2450.00 |    NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 |
3000.00 |    NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 |
5000.00 |    NULL |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 |
1500.00 |    0.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 |
1100.00 |    NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |
 950.00 |    NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 |
3000.00 |    NULL |     20 |
```

```
|  7934 | MILLER | CLERK     | 7782 | 1982-01-23 |
1300.00 |    NULL |     10 |
+-------+--------+-----------+------+------------+--
-------+---------+--------+

mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+

mysql> select * from salgrade;
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+
```

12.不看表中的数据，只看表的结构命令：

```
desc 表名;
mysql> desc dept;
+--------+-------------+------+-----+---------+-----
--+
```

```
| Field  | Type         | Null | Key | Default |
Extra |
+--------+-------------+------+-----+---------+-----
--+
| DEPTNO | int(2)      | NO   | PRI | NULL    |
  |部门编号
| DNAME  | varchar(14) | YES  |     | NULL    |
  |部门名字
| LOC    | varchar(13) | YES  |     | NULL    |
  |地理位置
+--------+-------------+------+-----+---------+-----
--+
mysql> desc emp;
+----------+-------------+------+-----+---------+---
-----+
| Field    | Type         | Null | Key | Default |
Extra |
+----------+-------------+------+-----+---------+---
-----+
| EMPNO    | int(4)      | NO   | PRI | NULL    |
    |员工编号
| ENAME    | varchar(10) | YES  |     | NULL    |
    |员工姓名
| JOB      | varchar(9)  | YES  |     | NULL    |
    |工作岗位
| MGR      | int(4)      | YES  |     | NULL    |
    |上级编号
| HIREDATE | date        | YES  |     | NULL    |
    |入职日期
| SAL      | double(7,2) | YES  |     | NULL    |
    |工资
| COMM     | double(7,2) | YES  |     | NULL    |
    |补助
| DEPTNO   | int(2)      | YES  |     | NULL    |
    |部门编号
+----------+-------------+------+-----+---------+---
-----+
```

```
mysql> desc salgrade;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| GRADE | int(11) | YES  |     | NULL    |       |工
资等级
| LOSAL | int(11) | YES  |     | NULL    |       |最
低工资
| HISAL | int(11) | YES  |     | NULL    |       |最
高工资
+-------+---------+------+-----+---------+-------+

describe缩写为：desc
mysql> describe dept;
+--------+-------------+------+-----+---------+-----
--+
| Field  | Type        | Null | Key | Default |
Extra |
+--------+-------------+------+-----+---------+-----
--+
| DEPTNO | int(2)      | NO   | PRI | NULL    |
  |
| DNAME  | varchar(14) | YES  |     | NULL    |
  |
| LOC    | varchar(13) | YES  |     | NULL    |
  |
+--------+-------------+------+-----+---------+-----
--+
```

13.DQL-简单查询

13.1、查询一个字段

```
select 字段名 from 表名;

其中要注意：
    select和from都是关键字。
    字段名和表名都是标识符。
```

强调：

对于SQL语句来说，是通用的，

所有的SQL语句以“;”结尾。

另外SQL语句不区分大小写，都行。

```
查询部门名字
mysql> select dname from dept;
+------------+
| dname      |
+------------+
| ACCOUNTING |
| RESEARCH   |
| SALES      |
| OPERATIONS |
+------------+
4 rows in set (0.00 sec)

mysql> SELECT DNAME FROM DEPT;
+------------+
| DNAME      |
+------------+
| ACCOUNTING |
| RESEARCH   |
| SALES      |
| OPERATIONS |
+------------+
4 rows in set (0.00 sec)
```

13.2、查询两个字段，或者多个字段怎么办

使用逗号隔开“,”
查询部门编号和部门名？

```
select deptno,dname from dept;
+--------+------------+
| deptno | dname      |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
```

13.3、查询所有字段怎么办？

第一种方式：可以把每个字段都写上

```
    select a,b,c,d,e,f... from tablename;
```

第二种方式：可以使用*

```
select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

这种方式的缺点：

1、效率低

2、可读性差。

在实际开发中不建议，可以自己玩没问题。

你可以在DOS命令窗口中想快速的看一看全表数据可以采用这种方式。

13.4、给查询的列起别名

```
mysql> select deptno,dname as deptname from dept;
+--------+------------+
| deptno | deptname   |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
```

使用as关键字起别名。

注意：只是将显示的查询结果列名显示为deptname，原表列名还是叫：dname

记住：select语句是永远都不会进行修改操作的。（因为只负责查询）

as关键字可以省略吗？可以的

```
mysql> select deptno,dname deptname from dept;
```

假设起别名的时候，别名里面有空格，怎么办？

```
mysql> select deptno,dname dept name from dept;
DBMS看到这样的语句，进行SQL语句的编译，不符合语法，编译报错。
怎么解决？
-select deptno,dname 'dept name' from dept; //加单引号
select deptno,dname "dept name" from dept; //加双引号
+--------+------------+
| deptno | dept name  |
+--------+------------+
|     10 | ACCOUNTING |
|     20 | RESEARCH   |
|     30 | SALES      |
|     40 | OPERATIONS |
+--------+------------+
```

注意：在所有的数据库当中，字符串统一使用单引号括起来，

单引号是标准，双引号在oracle数据库中用不了。但是在mysql

中可以使用。

再次强调：数据库中的字符串都是采用单引号括起来。这是标准的。

双引号不标准。

13.5、计算员工年薪?

sal * 12

```
mysql> select ename,sal from emp;
    +--------+---------+
    | ename  | sal     |
    +--------+---------+
    | SMITH  |  800.00 |
    | ALLEN  | 1600.00 |
    | WARD   | 1250.00 |
    | JONES  | 2975.00 |
    | MARTIN | 1250.00 |
```

```
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
| TURNER | 1500.00 |
| ADAMS  | 1100.00 |
| JAMES  |  950.00 |
| FORD   | 3000.00 |
| MILLER | 1300.00 |
+--------+---------+
mysql> select ename,sal*12 from emp; // 结论：字段可以使用数学表达式!
+--------+----------+
| ename  | sal*12   |
+--------+----------+
| SMITH  |  9600.00 |
| ALLEN  | 19200.00 |
| WARD   | 15000.00 |
| JONES  | 35700.00 |
| MARTIN | 15000.00 |
| BLAKE  | 34200.00 |
| CLARK  | 29400.00 |
| SCOTT  | 36000.00 |
| KING   | 60000.00 |
| TURNER | 18000.00 |
| ADAMS  | 13200.00 |
| JAMES  | 11400.00 |
| FORD   | 36000.00 |
| MILLER | 15600.00 |
+--------+----------+

mysql> select ename,sal*12 as yearsal from emp; //起别名
+--------+----------+
| ename  | yearsal  |
+--------+----------+
| SMITH  |  9600.00 |
```

```
| ALLEN  | 19200.00 |
| WARD   | 15000.00 |
| JONES  | 35700.00 |
| MARTIN | 15000.00 |
| BLAKE  | 34200.00 |
| CLARK  | 29400.00 |
| SCOTT  | 36000.00 |
| KING   | 60000.00 |
| TURNER | 18000.00 |
| ADAMS  | 13200.00 |
| FORD   | 36000.00 |
| MILLER | 15600.00 |
+--------+----------+
```

```
mysql> select ename,sal*12 as '年薪' from emp; //别名是中文，用单引号括起来。
+--------+----------+
| ename  | 年薪     |
+--------+----------+
| SMITH  |  9600.00 |
| ALLEN  | 19200.00 |
| WARD   | 15000.00 |
| JONES  | 35700.00 |
| MARTIN | 15000.00 |
| BLAKE  | 34200.00 |
| CLARK  | 29400.00 |
| SCOTT  | 36000.00 |
| KING   | 60000.00 |
| TURNER | 18000.00 |
| ADAMS  | 13200.00 |
| JAMES  | 11400.00 |
| FORD   | 36000.00 |
| MILLER | 15600.00 |
+--------+----------+
```

14、DQL-条件查询

14.1、什么是条件查询?

不是将表中所有数据都查出来。是查询出来符合条件的。
语法格式:

```
select
字段1,字段2,字段3....
from
表名
where
条件;
```

14.2、都有哪些条件?

```
= 等于
查询薪资等于800的员工姓名和编号?
    select empno,ename from emp where sal = 800;
查询SMITH的编号和薪资?
    select empno,sal from emp where ename = 'SMITH';
//字符串使用单引号



<>或!= 不等于
查询薪资不等于800的员工姓名和编号?
    select empno,ename from emp where sal != 800;
    select empno,ename from emp where sal <> 800; //
小于号和大于号组成的不等号



< 小于
查询薪资小于2000的员工姓名和编号?
    mysql> select empno,ename,sal from emp where sal
< 2000;
    +-------+--------+---------+
```

```
| empno | ename  | sal     |
+-------+--------+---------+
|  7369 | SMITH  |  800.00 |
|  7499 | ALLEN  | 1600.00 |
|  7521 | WARD   | 1250.00 |
|  7654 | MARTIN | 1250.00 |
|  7844 | TURNER | 1500.00 |
|  7876 | ADAMS  | 1100.00 |
|  7900 | JAMES  |  950.00 |
|  7934 | MILLER | 1300.00 |
+-------+--------+---------+
```

<= 小于等于
查询薪资小于等于3000的员工姓名和编号？

```
select empno,ename,sal from emp where sal <= 3000;
```

>大于
查询薪资大于3000的员工姓名和编号？

```
select empno,ename,sal from emp where sal > 3000;
```

>= 大于等于
查询薪资大于等于3000的员工姓名和编号？

```
select empno,ename,sal from emp where sal >= 3000;
```

between … and …. 两个值之间，等同于 >= and <=
查询薪资在2450和3000之间的员工信息？包括2450和3000

第一种方式：>= and <= （and是并且的意思。）

```
select empno,ename,sal from emp where sal >= 2450 and sal <= 3000;
+-------+-------+---------+
| empno | ename | sal     |
+-------+-------+---------+
|  7566 | JONES | 2975.00 |
|  7698 | BLAKE | 2850.00 |
|  7782 | CLARK | 2450.00 |
|  7788 | SCOTT | 3000.00 |
|  7902 | FORD  | 3000.00 |
+-------+-------+---------+
```

第二种方式：between … and …

```
select
    empno,ename,sal
from
    emp
where
    sal between 2450 and 3000;
```

注意：

使用between and的时候，必须遵循左小右大。

between and是闭区间，包括两端的值。

is null 为 null（is not null 不为空）

查询哪些员工的津贴/补助为null？

```
mysql> select empno,ename,sal,comm from emp where comm = null;
Empty set (0.00 sec)

mysql> select empno,ename,sal,comm from emp where comm is null;
+-------+--------+---------+------+
| empno | ename  | sal     | comm |
```

```
    +-------+--------+---------+------+ |  7369 |
SMITH  |  800.00 | NULL |
    |  7566 | JONES  | 2975.00 | NULL |
    |  7698 | BLAKE  | 2850.00 | NULL |
    |  7782 | CLARK  | 2450.00 | NULL |
    |  7788 | SCOTT  | 3000.00 | NULL |
    |  7839 | KING   | 5000.00 | NULL |
    |  7876 | ADAMS  | 1100.00 | NULL |
    |  7900 | JAMES  |  950.00 | NULL |
    |  7902 | FORD   | 3000.00 | NULL |
    |  7934 | MILLER | 1300.00 | NULL |
    +-------+--------+---------+------+
    10 rows in set (0.00 sec)
```

注意：在数据库当中null不能使用等号进行衡量。需要使用is null

因为数据库中的null代表什么也没有，它不是一个值，所以不能使用

等号衡量。

查询哪些员工的津贴/补助不为null？

```
    select empno,ename,sal,comm from emp where comm
is not null;
    +-------+--------+---------+---------+
    | empno | ename  | sal     | comm    |
    +-------+--------+---------+---------+
    |  7499 | ALLEN  | 1600.00 |  300.00 |
    |  7521 | WARD   | 1250.00 |  500.00 |
    |  7654 | MARTIN | 1250.00 | 1400.00 |
    |  7844 | TURNER | 1500.00 |    0.00 |
    +-------+--------+---------+---------+
```

and 并且

查询工作岗位是MANAGER并且工资大于2500的员工信息？

```
    select
        empno,ename,job,sal
```

```
    from
        emp
    where
        job = 'MANAGER' and sal > 2500;

    +-------+-------+---------+---------+
    | empno | ename | job     | sal     |
    +-------+-------+---------+---------+
    |  7566 | JONES | MANAGER | 2975.00 |
    |  7698 | BLAKE | MANAGER | 2850.00 |
    +-------+-------+---------+---------+
```

or 或者

查询工作岗位是MANAGER和SALESMAN的员工?

```
    select empno,ename,job from emp where job =
'MANAGER';
    select empno,ename,job from emp where job =
'SALESMAN';

    select
        empno,ename,job
    from
        emp
    where
        job = 'MANAGER' or job = 'SALESMAN';

    +-------+--------+----------+
    | empno | ename  | job      |
    +-------+--------+----------+
    |  7499 | ALLEN  | SALESMAN |
    |  7521 | WARD   | SALESMAN |
    |  7566 | JONES  | MANAGER  |
    |  7654 | MARTIN | SALESMAN |
    |  7698 | BLAKE  | MANAGER  |
    |  7782 | CLARK  | MANAGER  |
```

```
|  7844 | TURNER | SALESMAN |
+-------+--------+----------+
```

and和or同时出现的话，有优先级问题吗？

查询工资大于2500，并且部门编号为10或20部门的员工？

```
select
    *
from
    emp
where
    sal > 2500 and deptno = 10 or deptno = 20;
```

分析以上语句的问题？

and优先级比or高。

以上语句会先执行and，然后执行or。

以上这个语句表示什么含义？

找出工资大于2500并且部门编号为10的员工，或者20部门所有员工找出来。

```
select
    *
from
    emp
where
    sal > 2500 and (deptno = 10 or deptno = 20);
```

and和or同时出现，and优先级较高。如果想让or先执行，需要加“小括号”

★以后在开发中，如果不确定优先级，就加小括号就行了。

in 包含，相当于多个 or （not in 不在这个范围中） ！！不是区间

查询工作岗位是MANAGER和SALESMAN的员工？

```
        select empno,ename,job from emp where job =
'MANAGER' or job = 'SALESMAN';
        select empno,ename,job from emp where job
in('MANAGER', 'SALESMAN');
        +-------+--------+----------+
        | empno | ename  | job      |
        +-------+--------+----------+
        |  7499 | ALLEN  | SALESMAN |
        |  7521 | WARD   | SALESMAN |
        |  7566 | JONES  | MANAGER  |
        |  7654 | MARTIN | SALESMAN |
        |  7698 | BLAKE  | MANAGER  |
        |  7782 | CLARK  | MANAGER  |
        |  7844 | TURNER | SALESMAN |
        +-------+--------+----------+
        注意：in不是一个区间。in后面跟的是具体的值。

    查询薪资是800和5000的员工信息？
        select ename,sal from emp where sal = 800 or
sal = 5000;
        select ename,sal from emp where sal in(800,
5000); //这个不是表示800到5000都找出来。
        +-------+---------+
        | ename | sal     |
        +-------+---------+
        | SMITH |  800.00 |
        | KING  | 5000.00 |
        +-------+---------+
        select ename,sal from emp where sal in(800,
5000, 3000);

        // not in 表示不在这几个值当中的数据。
        select ename,sal from emp where sal not
in(800, 5000, 3000);
        +--------+---------+
        | ename  | sal     |
        +--------+---------+
```

```
        | ALLEN  | 1600.00 |
        | WARD   | 1250.00 |
        | JONES  | 2975.00 |
        | MARTIN | 1250.00 |
        | BLAKE  | 2850.00 |
        | CLARK  | 2450.00 |
        | TURNER | 1500.00 |
        | ADAMS  | 1100.00 |
        | JAMES  |  950.00 |
        | MILLER | 1300.00 |
        +--------+---------+
```

not 可以取非，主要用在 is 或 in 中

```
    is null
    is not null
    in
    not in
```

like

称为模糊查询，支持%或下划线匹配

%匹配任意多个字符

下划线：任意一个字符。

（%是一个特殊的符号，_ 也是一个特殊符号）

找出名字中含有O的？

```
    mysql> select ename from emp where ename like
'%O%';
    +-------+
    | ename |
    +-------+
    | JONES |    | SCOTT |
    | FORD  |
    +-------+
```

```
    找出名字以T结尾的？
        select ename from emp where ename like '%T';
            找出名字以K开始的？
        select ename from emp where ename like 'K%';

    找出第二个字每是A的？
        select ename from emp where ename like
'_A%';

    找出第三个字母是R的？
        select ename from emp where ename like
'__R%';

    t_student学生表
    name字段

----------------------

    zhangsan
    lisi
    wangwu
    zhaoliu
    jack_son

    找出名字中有“_”的？
        select name from t_student where name like
'%_%'; //这样不行。

        mysql> select name from t_student where name
like '%\_%'; // \转义字符。
        +----------+
        | name     |
        +----------+
        | jack_son |
        +----------+
```

15、排序

15.1、查询所有员工薪资，排序?

```
select
   ename,sal
from
   emp
order by
   sal; // 默认是升序！！
```

```
+--------+---------+
| ename  | sal     |
+--------+---------+
| SMITH  |  800.00 |
| JAMES  |  950.00 |
| ADAMS  | 1100.00 |
| WARD   | 1250.00 |
| MARTIN | 1250.00 |
| MILLER | 1300.00 |
| TURNER | 1500.00 |
| ALLEN  | 1600.00 |
| CLARK  | 2450.00 |
| BLAKE  | 2850.00 |
| JONES  | 2975.00 |
| FORD   | 3000.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
+--------+---------+
```

15.2、降序、升序

指定降序:

```
select
    ename,sal
from
    emp
order by
    sal desc;
```

```
+--------+---------+
| ename  | sal     |
+--------+---------+
| KING   | 5000.00 |
| SCOTT  | 3000.00 |
| FORD   | 3000.00 |
| JONES  | 2975.00 |
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| ALLEN  | 1600.00 |
| TURNER | 1500.00 |
| MILLER | 1300.00 |
| MARTIN | 1250.00 |
| WARD   | 1250.00 |
| ADAMS  | 1100.00 |
| JAMES  |  950.00 |
| SMITH  |  800.00 |
+--------+---------+
```

指定升序?

```
select
    ename,sal
from
    emp
order by
    sal asc;
```

```
+--------+---------+
```

```
| ename  | sal     |
+--------+---------+
| SMITH  |  800.00 |
| JAMES  |  950.00 |
| ADAMS  | 1100.00 |
| WARD   | 1250.00 |
| MARTIN | 1250.00 |
| MILLER | 1300.00 |
| TURNER | 1500.00 |
| ALLEN  | 1600.00 |
| CLARK  | 2450.00 |
| BLAKE  | 2850.00 |
| JONES  | 2975.00 |
| FORD   | 3000.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
+--------+---------+
```

15.3、两个字段排序、按照多个字段排序

查询员工名字和薪资，要求按照薪资升序，如果薪资一样的话，再按照名字升序排列。

```
    select
        ename,sal
    from
        emp
    order by
        sal asc, ename asc; // sal在前，起主导，只有sal相等的时候，才会考虑启用ename排序
```

```
+--------+---------+
| ename  | sal     |
+--------+---------+
| SMITH  |  800.00 |
| JAMES  |  950.00 |
```

```
| ADAMS  | 1100.00 |
| MARTIN | 1250.00 |
| WARD   | 1250.00 |
| MILLER | 1300.00 |
| TURNER | 1500.00 |
| ALLEN  | 1600.00 |
| CLARK  | 2450.00 |
| BLAKE  | 2850.00 |
| JONES  | 2975.00 |
| FORD   | 3000.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
+--------+---------+
```

15.4、了解：根据字段的位置也可以排序

```
select ename,sal from emp order by 2; // 2表示第二列。第二列是sal
```

按照查询结果的第2列sal排序。

了解一下，不建议在开发中这样写，因为不健壮。
因为列的顺序很容易发生改变，列顺序修改之后，2就废了。

16、综合一点的案例：

找出工资在1250到3000之间的员工信息，要求按照薪资降序排列。

```
    select
        ename,sal
    from
        emp
    where
        sal between 1250 and 3000
    order by
        sal desc;
```

```
+--------+---------+
| ename  | sal     |
+--------+---------+
| FORD   | 3000.00 |
| SCOTT  | 3000.00 |
| JONES  | 2975.00 |
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| ALLEN  | 1600.00 |
| TURNER | 1500.00 |
| MILLER | 1300.00 |
| MARTIN | 1250.00 |
| WARD   | 1250.00 |
+--------+---------+
```

关键字顺序不能变：

```
select
...
from
...
where
...
order by
...
```

以上语句的执行顺序必须掌握：
第一步：from
第二步：where
第三步：select
第四步：order by（排序总是在最后执行！）

17、数据处理函数

17.1、数据处理函数又被称为单行处理函数

单行处理函数的特点：一个输入对应一个输出。

和单行处理函数相对的是：多行处理函数。（多行处理函数特点：多个输入，对应1个输出！）

17.2、单行处理函数常见的有哪些?

```
lower 转换小写
    mysql> select lower(ename) as ename from emp;
    +--------+
    | ename  |
    +--------+
    | smith  |
    | allen  |
    | ward   |
    | jones  |
    | martin |
    | blake  |
    | clark  |
    | scott  |
    | king   |
    | turner |
    | adams  |
    | james  |
    | ford   |
```

```
    | miller |
    +--------+
```
14个输入，最后还是14个输出。这是单行处理函数的特点。

upper 转换大写

```
    mysql> select * from t_student;
    +----------+
    | name     |
    +----------+
    | zhangsan |
    | lisi     |
    | wangwu   |
    | jack_son |
    +----------+

    mysql> select upper(name) as name from
t_student;
    +----------+
    | name     |
    +----------+
    | ZHANGSAN |
    | LISI     |
    | WANGWU   |
    | JACK_SON |
    +----------+
```

substr 取子串（substr(被截取的字符串, 起始下标,截取的长度)）

```
    select substr(ename, 1, 1) as ename from emp;
```
注意：起始下标从1开始，没有0.

找出员工名字第一个字母是A的员工信息？

第一种方式：模糊查询

```
            select ename from emp where ename like
'A%';
```

第二种方式：substr函数

```
            select
                ename
            from
                emp
            where
                substr(ename,1,1) = 'A';
```

首字母大写？

```
        select name from t_student;
        select upper(substr(name,1,1)) from
t_student;
        select substr(name,2,length(name) - 1) from
t_student;
        select
concat(upper(substr(name,1,1)),substr(name,2,length(
name) - 1)) as result from t_student;
        +----------+
        | result   |
        +----------+
        | Zhangsan |
        | Lisi     |
        | Wangwu   |
        | Jack_son |
        +----------+
```

concat函数进行字符串的拼接

```
    select concat(empno,ename) from emp;
    +--------------------+
    | concat(empno,ename) |
    +--------------------+
    | 7369SMITH          |
```

```
    | 7499ALLEN            |
    | 7521WARD             |
    | 7566JONES            |
    | 7654MARTIN           |
    | 7698BLAKE            |
    | 7782CLARK            |
    | 7788SCOTT            |
    | 7839KING             |
    | 7844TURNER           |
    | 7876ADAMS            |
    | 7900JAMES            |
    | 7902FORD             |
    | 7934MILLER           |
    +----------------------+
```

length 取长度

```
    select length(ename) enamelength from emp;
    +-------------+
    | enamelength |
    +-------------+
    |           5 |
    |           5 |
    |           4 |
    |           5 |
    |           6 |
    |           5 |
    |           5 |
    |           5 |
    |           4 |
    |           6 |
    |           5 |
    |           5 |
    |           4 |
    |           6 |
```

```
    +-------------+




trim 去空格
    mysql> select * from emp where ename = '  KING';
    Empty set (0.00 sec)

    mysql> select * from emp where ename = trim('
KING');
    +-------+-------+-----------+------+-----------
+---------+------+--------+
    | EMPNO | ENAME | JOB       | MGR  | HIREDATE
| SAL     | COMM | DEPTNO |
    +-------+-------+-----------+------+-----------
+---------+------+--------+
    |  7839 | KING  | PRESIDENT | NULL | 1981-11-17
| 5000.00 | NULL |     10 |
    +-------+-------+-----------+------+-----------
+---------+------+--------+

str_to_date 将字符串转换成日期
date_format 格式化日期
format 设置千分位

case..when..then..when..then..else..end
    当员工的工作岗位是MANAGER的时候，工资上调10%，当工作岗位是SALESMAN的时候，工资上调50%,其它正常。
    （注意：不修改数据库，只是将查询结果显示为工资上调）
    select
        ename,
        job,
        sal as oldsal,
        (case job when 'MANAGER' then sal*1.1 when
'SALESMAN' then sal*1.5 else sal end) as newsal
    from
```

```
    emp;

+--------+-----------+---------+---------+
| ename  | job       | oldsal  | newsal  |
+--------+-----------+---------+---------+
| SMITH  | CLERK     |  800.00 |  800.00 |
| ALLEN  | SALESMAN  | 1600.00 | 2400.00 |
| WARD   | SALESMAN  | 1250.00 | 1875.00 |
| JONES  | MANAGER   | 2975.00 | 3272.50 |
| MARTIN | SALESMAN  | 1250.00 | 1875.00 |
| BLAKE  | MANAGER   | 2850.00 | 3135.00 |
| CLARK  | MANAGER   | 2450.00 | 2695.00 |
| SCOTT  | ANALYST   | 3000.00 | 3000.00 |
| KING   | PRESIDENT | 5000.00 | 5000.00 |
| TURNER | SALESMAN  | 1500.00 | 2250.00 |
| ADAMS  | CLERK     | 1100.00 | 1100.00 |
| JAMES  | CLERK     |  950.00 |  950.00 |
| FORD   | ANALYST   | 3000.00 | 3000.00 |
| MILLER | CLERK     | 1300.00 | 1300.00 |
+--------+-----------+---------+---------+
```

round 四舍五入

```
    select 字段 from 表名;
    select ename from emp;
    select 'abc' from emp; // select后面直接跟“字面量/字面值”

    mysql> select 'abc' as bieming from emp;
    +---------+
    | bieming |
    +---------+
    | abc     |
    | abc     |
    | abc     |
```

```
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
| abc     |
+---------+

mysql> select abc from emp;
ERROR 1054 (42S22): Unknown column 'abc' in
'field list'
```

这样肯定报错，因为会把abc当做一个字段的名字，去emp表中找abc字段去了。

```
select 1000 as num from emp; // 1000 也是被当做一个
```

字面量/字面值。

```
+------+
| num  |
+------+
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
| 1000 |
```

```
| 1000 |
+------+
```

结论：select后面可以跟某个表的字段名（可以等同看做变量名），也可以跟字面量/字面值（数据）。

```
select 21000 as num from dept;
+-------+
| num   |
+-------+
| 21000 |
| 21000 |
| 21000 |
| 21000 |
+-------+
```

```
mysql> select round(1236.567, 0) as result from emp; //保留整数位。
+--------+
| result |
+--------+
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
|   1237 |
+--------+
```

```
    select round(1236.567, 1) as result from emp; // 保留1个小数
    select round(1236.567, 2) as result from emp; // 保留2个小数
    select round(1236.567, -1) as result from emp; // 保留到十位。
    +--------+
    | result |
    +--------+
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    |   1240 |
    +--------+

    select round(1236.567, -2) as result from emp;
    +--------+
    | result |
    +--------+
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
```

```
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    |   1200 |
    +--------+
```

rand() 生成随机数

```
    mysql> select round(rand()*100,0) from emp; //
```

100以内的随机数

```
    +---------------------+
    | round(rand()*100,0) |
    +---------------------+
    |                  76 |
    |                  29 |
    |                  15 |
    |                  88 |
    |                  95 |
    |                   9 |
    |                  63 |
    |                  89 |
    |                  54 |
    |                   3 |
    |                  54 |
    |                  61 |
    |                  42 |
    |                  28 |
    +---------------------+
```

ifnull 可以将 null 转换成一个具体值

ifnull是空处理函数。专门处理空的。

在所有数据库当中，只要有NULL参与的数学运算，最终结果就是NULL。

```
mysql> select ename, sal + comm as salcomm from emp;
+--------+---------+
| ename  | salcomm |
+--------+---------+
| SMITH  |    NULL |
| ALLEN  | 1900.00 |
| WARD   | 1750.00 |
| JONES  |    NULL |
| MARTIN | 2650.00 |
| BLAKE  |    NULL |
| CLARK  |    NULL |
| SCOTT  |    NULL |
| KING   |    NULL |
| TURNER | 1500.00 |
| ADAMS  |    NULL |
| JAMES  |    NULL |
| FORD   |    NULL |
| MILLER |    NULL |
+--------+---------+
```

计算每个员工的年薪？

年薪 = (月薪 + 月补助) * 12

```
select ename, (sal + comm) * 12 as yearsal from emp;
+--------+----------+
| ename  | yearsal  |
+--------+----------+
| SMITH  |     NULL |
| ALLEN  | 22800.00 |
| WARD   | 21000.00 |
| JONES  |     NULL |
| MARTIN | 31800.00 |
```

```
| BLAKE  |     NULL |
| CLARK  |     NULL |
| SCOTT  |     NULL |
| KING   |     NULL |
| TURNER | 18000.00 |
| ADAMS  |     NULL |
| JAMES  |     NULL |
| FORD   |     NULL |
| MILLER |     NULL |
+--------+----------+
```

注意：NULL只要参与运算，最终结果一定是NULL。为了避免这个现象，需要使用ifnull函数。

ifnull函数用法：ifnull(数据, 被当做哪个值)

如果“数据”为NULL的时候，把这个数据结构当做哪个值。

补助为NULL的时候，将补助当做0

```
select ename, (sal + ifnull(comm, 0)) * 12 as yearsal from emp;
```

```
+--------+----------+
| ename  | yearsal  |
+--------+----------+
| SMITH  |  9600.00 |
| ALLEN  | 22800.00 |
| WARD   | 21000.00 |
| JONES  | 35700.00 |
| MARTIN | 31800.00 |
| BLAKE  | 34200.00 |
| CLARK  | 29400.00 |
| SCOTT  | 36000.00 |
| KING   | 60000.00 |
| TURNER | 18000.00 |
| ADAMS  | 13200.00 |
| JAMES  | 11400.00 |
| FORD   | 36000.00 |
| MILLER | 15600.00 |
```

```
+--------+----------+
```

18、分组函数（多行处理函数）

多行处理函数的特点：输入多行，最终输出一行。

5个：

```
count    计数
sum 求和
avg 平均值
max 最大值
min 最小值
```

注意：

```
分组函数在使用的时候必须先进行分组，然后才能用。
如果你没有对数据进行分组，整张表默认为一组。
```

找出最高工资？

```
mysql> select max(sal) from emp;
+----------+
| max(sal) |
+----------+
|  5000.00 |
+----------+
```

找出最低工资？

```
mysql> select min(sal) from emp;
+----------+
| min(sal) |
+----------+
|   800.00 |
+----------+
```

计算工资和：

```
mysql> select sum(sal) from emp;
+----------+
| sum(sal) |
+----------+
| 29025.00 |
+----------+
```

计算平均工资：

```
mysql> select avg(sal) from emp;
+-------------+
| avg(sal)    |
+-------------+
| 2073.214286 |
+-------------+
```

14个工资全部加起来，然后除以14。

计算员工数量？

```
mysql> select count(ename) from emp;
+--------------+
| count(ename) |
+--------------+
|           14 |
+--------------+
```

分组函数在使用的时候需要注意哪些？

第一点：分组函数自动忽略NULL，你不需要提前对NULL进行处理。

```
mysql> select sum(comm) from emp;
+-----------+
| sum(comm) |
+-----------+
|   2200.00 |
+-----------+

mysql> select count(comm) from emp;
+-------------+
| count(comm) |
+-------------+
|           4 |
+-------------+
mysql> select avg(comm) from emp;
+-------------+
| avg(comm)   |
+-------------+
| 550.000000 |
+-------------+
```

第二点：分组函数中count(*)和count(具体字段)有什么区别？

```
mysql> select count(*) from emp;
+----------+
| count(*) |
+----------+
|       14 |
+----------+

mysql> select count(comm) from emp;
+-------------+
| count(comm) |
+-------------+
|           4 |
+-------------+
```

count(具体字段)：表示统计该字段下所有不为NULL的元素的总数。

count(*)：统计表当中的总行数。（只要有一行数据count则++）

因为每一行记录不可能都为NULL，一行数据中有一列不为NULL，则这行数据就是有效的。

第三点：分组函数不能够直接使用在where子句中。

找出比最低工资高的员工信息。

```
select ename,sal from emp where sal > min(sal);
```

表面上没问题，运行一下？

```
ERROR 1111 (HY000): Invalid use of group function
```

??

说完分组查询(group by)之后就明白了了。

第四点：所有的分组函数可以组合起来一起用。

```
select sum(sal),min(sal),max(sal),avg(sal),count(*) from emp;
+---------+---------+---------+-------------+----------+
| sum(sal) | min(sal) | max(sal) | avg(sal)    | count(*) |
+---------+---------+---------+-------------+----------+
| 29025.00 |   800.00 |  5000.00 | 2073.214286 |       14 |
+---------+---------+---------+-------------+----------+
```

★★★19、分组查询（非常重要：五颗星*）

19.1、什么是分组查询

在实际的应用中，可能有这样的需求，需要先进行分组，然后对每一组的数据进行操作。

这个时候我们需要使用分组查询，怎么进行分组查询呢？

```
select
    ...
from
    ...
group by
    ...
计算每个部门的工资和？
计算每个工作岗位的平均薪资？
找出每个工作岗位的最高薪资？
....
```

19.2、将之前的关键字全部组合在一起，来看一下他们的执行顺序

```
select
    ...
from
    ...
where
    ...
group by
    ...
order by
    ...
```

以上关键字的顺序不能颠倒，需要记忆。
执行顺序是什么

1.from
2. where
3. group by
4. select
5. order by

为什么分组函数不能直接使用在where后面?

```
select ename,sal from emp where sal > min(sal);//报错。
```

因为**分组函数在使用的时候必须先分组之后才能使用。**
where执行的时候，还没有分组。所以where后面不能出现分组函数。

```
select sum(sal) from emp;
```

这个没有分组，为啥sum()函数可以用呢？因为select在group by之后执行。

19.3、找出每个工作岗位的工资和

实现思路：按照工作岗位分组，然后对工资求和。

```
    select
            job,sum(sal)
        from
            emp
        group by
            job;

        +-----------+----------+
```

```
| job       | sum(sal) |
+-----------+----------+
| ANALYST   |  6000.00 |
| CLERK     |  4150.00 |
| MANAGER   |  8275.00 |
| PRESIDENT |  5000.00 |
| SALESMAN  |  5600.00 |
+-----------+----------+
```

以上这个语句的执行顺序?

先从emp表中查询数据。

根据job字段进行分组。

然后对每一组的数据进行sum(sal)

```
select ename,job,sum(sal) from emp group by job;
+-------+-----------+----------+
| ename | job       | sum(sal) |
+-------+-----------+----------+
| SCOTT | ANALYST   |  6000.00 |
| SMITH | CLERK     |  4150.00 |
| JONES | MANAGER   |  8275.00 |
| KING  | PRESIDENT |  5000.00 |
| ALLEN | SALESMAN  |  5600.00 |
+-------+-----------+----------+
```

以上语句在mysql中可以执行，但是毫无意义。

以上语句在oracle中执行报错。

oracle的语法比mysql的语法严格。（mysql的语法相对来说松散一些！）

重点结论：

在一条select语句当中，如果有group by语句的话，

select后面只能跟：参加分组的字段，以及分组函数。

其它的一律不能跟。

19.4、找出每个部门的最高薪资

实现思路是什么？

按照部门编号分组，求每一组的最大值。

select后面添加ename字段没有意义，另外oracle会报错。

```
    mysql> select ename,deptno,max(sal) from emp
group by deptno;
        +-------+--------+----------+
        | ename | deptno | max(sal) |
        +-------+--------+----------+
        | CLARK |     10 |  5000.00 |
        | SMITH |     20 |  3000.00 |
        | ALLEN |     30 |  2850.00 |
        +-------+--------+----------+

        mysql> select deptno,max(sal) from emp group
by deptno;
        +--------+----------+
        | deptno | max(sal) |
        +--------+----------+
        |     10 |  5000.00 |
        |     20 |  3000.00 |
        |     30 |  2850.00 |
        +--------+----------+
```

19.5、找出“每个部门，不同工作岗位”的最高薪资

```
    +--------+-----------+---------+--------+
    | ename  | job       | sal     | deptno |
    +--------+-----------+---------+--------+
    | MILLER | CLERK     | 1300.00 |     10 |
    | KING   | PRESIDENT | 5000.00 |     10 |
```

```
| CLARK  | MANAGER   | 2450.00 |     10 |

| FORD   | ANALYST   | 3000.00 |     20 |
| ADAMS  | CLERK     | 1100.00 |     20 |
| SCOTT  | ANALYST   | 3000.00 |     20 |
| JONES  | MANAGER   | 2975.00 |     20 |

| BLAKE  | MANAGER   | 2850.00 |     30 |
| MARTIN | SALESMAN  | 1250.00 |     30 |
| ALLEN  | SALESMAN  | 1600.00 |     30 |
| TURNER | SALESMAN  | 1500.00 |     30 |
| WARD   | SALESMAN  | 1250.00 |     30 |
| JAMES  | CLERK     |  950.00 |     30 |
+--------+-----------+---------+--------+
```

技巧：两个字段联合成1个字段看。（两个字段联合分组）

```
select
    deptno, job, max(sal)
from
    emp
group by
    deptno, job;

+--------+-----------+----------+
| deptno | job       | max(sal) |
+--------+-----------+----------+
|     10 | CLERK     |  1300.00 |
|     10 | MANAGER   |  2450.00 |
|     10 | PRESIDENT |  5000.00 |
|     20 | ANALYST   |  3000.00 |
|     20 | CLERK     |  1100.00 |
|     20 | MANAGER   |  2975.00 |
|     30 | CLERK     |   950.00 |
|     30 | MANAGER   |  2850.00 |
|     30 | SALESMAN  |  1600.00 |
+--------+-----------+----------+
```

19.6、使用having可以对分完组之后的数据进一步过滤。

having不能单独使用，having不能代替where，having必须和group by联合使用。

找出每个部门最高薪资，要求显示最高薪资大于3000的

第一步：找出每个部门最高薪资

按照部门编号分组，求每一组最大值。

```
select deptno,max(sal) from emp group by deptno;
+--------+----------+
| deptno | max(sal) |
+--------+----------+
|     10 |  5000.00 |
|     20 |  3000.00 |
|     30 |  2850.00 |
+--------+----------+
```

第二步：要求显示最高薪资大于3000

```
select
deptno,max(sal)
from
emp
group by
deptno
having
max(sal) > 3000;

+--------+----------+
| deptno | max(sal) |
+--------+----------+
|     10 |  5000.00 |
```

```
+--------+----------+
```

思考一个问题：以上的sql语句执行效率是不是低?

比较低，实际上可以这样考虑：先将大于3000的都找出来，然后再分组。

```
select
deptno,max(sal)
from
emp
where
sal > 3000
group by
deptno;

+--------+----------+
| deptno | max(sal) |
+--------+----------+
|     10 |  5000.00 |
+--------+----------+
```

优化策略：

where和having，优先选择where，where实在完成不了了，再选择having。

19.7、where没办法的

找出每个部门平均薪资，要求显示平均薪资高于2500的。

第一步：找出每个部门平均薪资

```
select deptno,avg(sal) from emp group by deptno;
+--------+-------------+
| deptno | avg(sal)    |
+--------+-------------+
|     10 | 2916.666667 |
|     20 | 2175.000000 |
|     30 | 1566.666667 |
+--------+-------------+
```

第二步：要求显示平均薪资高于2500的

```
select
deptno,avg(sal)
from
emp
group by
deptno
having
avg(sal) > 2500;

+--------+-------------+
| deptno | avg(sal)    |
+--------+-------------+
|     10 | 2916.666667 |
+--------+-------------+
```

20、大总结（单表的查询学完了）

```
select
...
from
...
where
...
group by
...
having
...
order by
...
```

以上关键字只能按照这个顺序来，不能颠倒。

执行顺序?

1. **from**
2. **where**
3. **group by**
4. **having**
5. **select**
6. **order by**

从某张表中查询数据，
先经过where条件筛选出有价值的数据。
对这些有价值的数据进行分组。
分组之后可以使用having继续筛选。
select查询出来。
最后排序输出！

找出每个岗位的平均薪资，要求显示平均薪资大于1500的，除MANAGER岗位之外，
要求按照平均薪资降序排。

```
    select
        job, avg(sal) as avgsal
```

```
    from
        emp
    where
        job <> 'MANAGER'
    group by
        job
    having
        avg(sal) > 1500
    order by
        avgsal desc;

    +-----------+-------------+
    | job       | avgsal      |
    +-----------+-------------+
    | PRESIDENT | 5000.000000 |
    | ANALYST   | 3000.000000 |
    +-----------+-------------+
```

day02

1、把查询结果去除重复记录【distinct】

注意：原表数据不会被修改，只是查询结果去重。
去重需要使用一个关键字：distinct

```
mysql> select distinct job from emp;
+-----------+
| job       |
+-----------+
| CLERK     |
| SALESMAN  |
```

```
| MANAGER   |
| ANALYST   |
| PRESIDENT |
+-----------+

// 以下编写是错误的，语法错误。
// distinct只能出现在所有字段的最前方。
mysql> select ename,distinct job from emp;

// distinct出现在job,deptno两个字段之前，表示两个字段联合起来去重。
mysql> select distinct job,deptno from emp;
+-----------+--------+
| job       | deptno |
+-----------+--------+
| CLERK     |     20 |
| SALESMAN  |     30 |
| MANAGER   |     20 |
| MANAGER   |     30 |
| MANAGER   |     10 |
| ANALYST   |     20 |
| PRESIDENT |     10 |
| CLERK     |     30 |
| CLERK     |     10 |
+-----------+--------+

统计一下工作岗位的数量？
    select count(distinct job) from emp;
    +--------------------+
    | count(distinct job) |
    +--------------------+
    |                  5 |
    +--------------------+
```

★★★2、连接查询

2.1、什么是连接查询

从一张表中单独查询，称为单表查询。

emp表和dept表联合起来查询数据，从emp表中取员工名字，从dept表中取部门名字。

这种跨表查询，多张表联合起来查询数据，被称为连接查询。

2.2、连接查询的分类?

```
根据语法的年代分类：
    SQL92：1992年的时候出现的语法
    SQL99：1999年的时候出现的语法
    我们这里重点学习SQL99.(这个过程中简单演示一个SQL92的例子)

根据表连接的方式分类：
    内连接：
        等值连接
        非等值连接
        自连接
    外连接：
        左外连接（左连接）
        右外连接（右连接）

    全连接（不讲）
```

2.3、当两张表进行连接查询时，没有任何条件的限制会发生什么现象

```
案例：查询每个员工所在部门名称？
    mysql> select ename,deptno from emp;
    +--------+--------+
    | ename  | deptno |
    +--------+--------+
    | SMITH  |     20 |
```

```
| ALLEN  |     30 |
| WARD   |     30 |
| JONES  |     20 |
| MARTIN |     30 |
| BLAKE  |     30 |
| CLARK  |     10 |
| SCOTT  |     20 |
| KING   |     10 |
| TURNER |     30 |
| ADAMS  |     20 |
| JAMES  |     30 |
| FORD   |     20 |
| MILLER |     10 |
+--------+--------+
mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

两张表连接没有任何条件限制：

```
select ename,dname from emp, dept;
+--------+------------+
| ename  | dname      |
+--------+------------+
| SMITH  | ACCOUNTING |
| SMITH  | RESEARCH   |
| SMITH  | SALES      |
| SMITH  | OPERATIONS |
| ALLEN  | ACCOUNTING |
| ALLEN  | RESEARCH   |
| ALLEN  | SALES      |
| ALLEN  | OPERATIONS |
```

```
    ...
    56 rows in set (0.00 sec)
    14 * 4 = 56
```

当两张表进行连接查询，没有任何条件限制的时候，最终查询结果条数，是
两张表条数的乘积，这种现象被称为：笛卡尔积现象。（笛卡尔发现的，这是
一个数学现象。）

2.4、怎么避免笛卡尔积现象?

连接时加条件，满足这个条件的记录被筛选出来!

```
select
ename,dname
from
emp, dept
where
emp.deptno = dept.deptno;
```

```
select
    emp.ename,dept.dname
from
    emp, dept
where
    emp.deptno = dept.deptno;

// 表起别名。很重要。效率问题。
select
    e.ename,d.dname
from
    emp e, dept d
where
    e.deptno = d.deptno; //SQL92语法。

+--------+------------+
```

```
| ename  | dname      |
+--------+------------+
| CLARK  | ACCOUNTING |
| KING   | ACCOUNTING |
| MILLER | ACCOUNTING |
| SMITH  | RESEARCH   |
| JONES  | RESEARCH   |
| SCOTT  | RESEARCH   |
| ADAMS  | RESEARCH   |
| FORD   | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| TURNER | SALES      |
| JAMES  | SALES      |
+--------+------------+
```

思考：最终查询的结果条数是14条，但是匹配的过程中，匹配的次数减少了吗？

还是56次，只不过进行了四选一。次数没有减少。

注意：通过笛卡尔积现象得出，表的连接次数越多效率越低，尽量避免表的
连接次数。

2.5、内连接之等值连接

案例：查询每个员工所在部门名称，显示员工名和部门名
emp e和dept d表进行连接。条件是：e.deptno = d.deptno

SQL92语法：

```
select
e.ename,d.dname
from
emp e, dept d
where
e.deptno = d.deptno;
```

sql92的缺点：结构不清晰，表的连接条件，和后期进一步筛选的条件，都放到了where后面。

SQL99语法：

```
select
e.ename,d.dname
from
emp e
join
dept d
on
e.deptno = d.deptno;
```

```
//inner可以省略（带着inner可读性更好！！！一眼就能看出来是内连接）
select
    e.ename,d.dname
from
    emp e
inner join
    dept d
on
    e.deptno = d.deptno; // 条件是等量关系，所以被称为等值连接。
```

sql99优点：表连接的条件是独立的，连接之后，如果还需要进一步筛选，再往后继续添加where

```
SQL99语法：
    select
        ...
    from
        a
    join
        b
    on
        a和b的连接条件
    where
        筛选条件
```

2.6、内连接之非等值连接

案例：找出每个员工的薪资等级，要求显示员工名、薪资、薪资等级

```
mysql> select * from emp; e
+-------+--------+-----------+------+------------+---
-------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   |
SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---
-------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |
 800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 |
1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 |
1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 |
2975.00 |    NULL |     20 |
....

mysql> select * from salgrade; s
+-------+-------+-------+
```

```
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+
```

```
select
    e.ename, e.sal, s.grade
from
    emp e
join
    salgrade s
on
    e.sal between s.losal and s.hisal; // 条件不是一个等量关系，称为非等值连接。
```

```
select
    e.ename, e.sal, s.grade
from
    emp e
inner join
    salgrade s
on
    e.sal between s.losal and s.hisal;
```

```
+--------+---------+-------+
| ename  | sal     | grade |
+--------+---------+-------+
| SMITH  |  800.00 |     1 |
| ALLEN  | 1600.00 |     3 |
| WARD   | 1250.00 |     2 |
| JONES  | 2975.00 |     4 |
```

```
| MARTIN | 1250.00 |     2 |
| BLAKE  | 2850.00 |     4 |
| CLARK  | 2450.00 |     4 |
| SCOTT  | 3000.00 |     4 |
| KING   | 5000.00 |     5 |
| TURNER | 1500.00 |     3 |
| ADAMS  | 1100.00 |     1 |
| JAMES  |  950.00 |     1 |
| FORD   | 3000.00 |     4 |
| MILLER | 1300.00 |     2 |
+--------+---------+-------+
```

2.7、内连接之自连接

案例：查询员工的上级领导，要求显示员工名和对应的领导名？

```
mysql> select empno,ename,mgr from emp;
+-------+--------+------+
| empno | ename  | mgr  |
+-------+--------+------+
|  7369 | SMITH  | 7902 |
|  7499 | ALLEN  | 7698 |
|  7521 | WARD   | 7698 |
|  7566 | JONES  | 7839 |
|  7654 | MARTIN | 7698 |
|  7698 | BLAKE  | 7839 |
|  7782 | CLARK  | 7839 |
|  7788 | SCOTT  | 7566 |
|  7839 | KING   | NULL |
|  7844 | TURNER | 7698 |
|  7876 | ADAMS  | 7788 |
|  7900 | JAMES  | 7698 |
|  7902 | FORD   | 7566 |
|  7934 | MILLER | 7782 |
+-------+--------+------+
```

技巧：一张表看成两张表。

emp a 员工表

empno	ename	mgr
7369	SMITH	7902
7499	ALLEN	7698
7521	WARD	7698
7566	JONES	7839
7654	MARTIN	7698
7698	BLAKE	7839
7782	CLARK	7839
7788	SCOTT	7566
7839	KING	NULL
7844	TURNER	7698
7876	ADAMS	7788
7900	JAMES	7698
7902	FORD	7566
7934	MILLER	7782

emp b 领导表

empno	ename	mgr
7369	SMITH	7902
7499	ALLEN	7698
7521	WARD	7698
7566	JONES	7839
7654	MARTIN	7698
7698	BLAKE	7839
7782	CLARK	7839
7788	SCOTT	7566
7839	KING	NULL
7844	TURNER	7698
7876	ADAMS	7788

```
|  7900 | JAMES  | 7698 |
|  7902 | FORD   | 7566 |
|  7934 | MILLER | 7782 |
+-------+--------+------+

select
    a.ename as '员工名', b.ename as '领导名'
from
    emp a
join
    emp b
on
    a.mgr = b.empno; //员工的领导编号 = 领导的员工编号

+--------+--------+
| 员工名 | 领导名|
+--------+--------+
| SMITH  | FORD   |
| ALLEN  | BLAKE  |
| WARD   | BLAKE  |
| JONES  | KING   |
| MARTIN | BLAKE  |
| BLAKE  | KING   |
| CLARK  | KING   |
| SCOTT  | JONES  |
| TURNER | BLAKE  |
| ADAMS  | SCOTT  |
| JAMES  | BLAKE  |
| FORD   | JONES  |
| MILLER | CLARK  |
+--------+--------+
```

13条记录，没有KING。《内连接》

以上就是内连接中的：自连接，技巧：一张表看做两张表。

2.8、外连接

```
mysql> select * from emp; e
+-------+--------+-----------+------+------------+--
-------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   |
SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+--
-------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |
 800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 |
1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 |
1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 |
2975.00 |    NULL |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 |
1250.00 | 1400.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 |
2850.00 |    NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 |
2450.00 |    NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 |
3000.00 |    NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 |
5000.00 |    NULL |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 |
1500.00 |    0.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 |
1100.00 |    NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |
 950.00 |    NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 |
3000.00 |    NULL |     20 |
```

```
|  7934 | MILLER | CLERK     | 7782 | 1982-01-23 |
1300.00 |    NULL |     10 |
+-------+--------+-----------+------+------------+--
-------+---------+--------+

mysql> select * from dept; d
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

内连接：（A和B连接，AB两张表没有主次关系。平等的。）

```
select
    e.ename,d.dname
from
    emp e
join
    dept d
on
    e.deptno = d.deptno; //内连接的特点：完成能够匹配上这个条件的数据查询出来。
```

```
+--------+------------+
| ename  | dname      |
+--------+------------+
| CLARK  | ACCOUNTING |
| KING   | ACCOUNTING |
| MILLER | ACCOUNTING |
| SMITH  | RESEARCH   |
| JONES  | RESEARCH   |
| SCOTT  | RESEARCH   |
| ADAMS  | RESEARCH   |
| FORD   | RESEARCH   |
```

```
| ALLEN  | SALES      |
| WARD   | SALES      |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| TURNER | SALES      |
| JAMES  | SALES      |
+--------+------------+
```

外连接（右外连接）：

```
select
    e.ename,d.dname
from
    emp e
right join
    dept d
on
    e.deptno = d.deptno;

// outer是可以省略的，带着可读性强。
select
    e.ename,d.dname
from
    emp e
right outer join
    dept d
on
    e.deptno = d.deptno;
```

right代表什么：表示将join关键字右边的这张表看成主表，主要是为了将
这张表的数据全部查询出来，捎带着关联查询左边的表。
在外连接当中，两张表连接，产生了主次关系。

外连接（左外连接）：

```
select
    e.ename,d.dname
from
```

```
    dept d
left join
    emp e
on
    e.deptno = d.deptno;

// outer是可以省略的，带着可读性强。
select
    e.ename,d.dname
from
    dept d
left outer join
    emp e
on
    e.deptno = d.deptno;
```

带有right的是右外连接，又叫做右连接。
带有left的是左外连接，又叫做左连接。
任何一个右连接都有左连接的写法。
任何一个左连接都有右连接的写法。

```
+--------+------------+
| ename  | dname      |
+--------+------------+
| CLARK  | ACCOUNTING |
| KING   | ACCOUNTING |
| MILLER | ACCOUNTING |
| SMITH  | RESEARCH   |
| JONES  | RESEARCH   |
| SCOTT  | RESEARCH   |
| ADAMS  | RESEARCH   |
| FORD   | RESEARCH   |
| ALLEN  | SALES      |
| WARD   | SALES      |
| MARTIN | SALES      |
| BLAKE  | SALES      |
| TURNER | SALES      |
```

```
| JAMES  | SALES      |
| NULL   | OPERATIONS |
+--------+------------+
```

思考：外连接的查询结果条数一定是 >= 内连接的查询结果条数？

正确。

案例：查询每个员工的上级领导，要求显示所有员工的名字和领导名？

```
select
a.ename as '员工名', b.ename as '领导名'
from
emp a
left join
emp b
on
a.mgr = b.empno;

+--------+--------+
| 员工名      | 领导名     |
+--------+--------+
| SMITH  | FORD   |
| ALLEN  | BLAKE  |
| WARD   | BLAKE  |
| JONES  | KING   |
| MARTIN | BLAKE  |
| BLAKE  | KING   |
| CLARK  | KING   |
| SCOTT  | JONES  |
| KING   | NULL   |
| TURNER | BLAKE  |
| ADAMS  | SCOTT  |
| JAMES  | BLAKE  |
| FORD   | JONES  |
| MILLER | CLARK  |
+--------+--------+
```

2.9、三张表，四张表怎么连接?

语法:

```
select
...
from
a
join
b
on
a和b的连接条件
join
c
on
a和c的连接条件
right join
d
on
a和d的连接条件
```

一条SQL中内连接和外连接可以混合。都可以出现!

案例：找出每个员工的部门名称以及工资等级，
要求显示员工名、部门名、薪资、薪资等级?

```
select
    e.ename,e.sal,d.dname,s.grade
from
    emp e
join
    dept d
on
    e.deptno = d.deptno
join
```

```
    salgrade s
on
    e.sal between s.losal and s.hisal;

+--------+---------+------------+-------+
| ename  | sal     | dname      | grade |
+--------+---------+------------+-------+
| SMITH  |  800.00 | RESEARCH   |     1 |
| ALLEN  | 1600.00 | SALES      |     3 |
| WARD   | 1250.00 | SALES      |     2 |
| JONES  | 2975.00 | RESEARCH   |     4 |
| MARTIN | 1250.00 | SALES      |     2 |
| BLAKE  | 2850.00 | SALES      |     4 |
| CLARK  | 2450.00 | ACCOUNTING |     4 |
| SCOTT  | 3000.00 | RESEARCH   |     4 |
| KING   | 5000.00 | ACCOUNTING |     5 |
| TURNER | 1500.00 | SALES      |     3 |
| ADAMS  | 1100.00 | RESEARCH   |     1 |
| JAMES  |  950.00 | SALES      |     1 |
| FORD   | 3000.00 | RESEARCH   |     4 |
| MILLER | 1300.00 | ACCOUNTING |     2 |
+--------+---------+------------+-------+
```

案例：找出每个员工的部门名称以及工资等级，还有上级领导，要求显示员工名、领导名、部门名、薪资、薪资等级？

```
select
    e.ename,e.sal,d.dname,s.grade,l.ename
from
    emp e
join
    dept d
on
    e.deptno = d.deptno
join
    salgrade s
on
```

```
    e.sal between s.losal and s.hisal
left join
    emp l
on
    e.mgr = l.empno;
```

```
+--------+---------+------------+-------+-------+
| ename  | sal     | dname      | grade | ename |
+--------+---------+------------+-------+-------+
| SMITH  |  800.00 | RESEARCH   |     1 | FORD  |
| ALLEN  | 1600.00 | SALES      |     3 | BLAKE |
| WARD   | 1250.00 | SALES      |     2 | BLAKE |
| JONES  | 2975.00 | RESEARCH   |     4 | KING  |
| MARTIN | 1250.00 | SALES      |     2 | BLAKE |
| BLAKE  | 2850.00 | SALES      |     4 | KING  |
| CLARK  | 2450.00 | ACCOUNTING |     4 | KING  |
| SCOTT  | 3000.00 | RESEARCH   |     4 | JONES |
| KING   | 5000.00 | ACCOUNTING |     5 | NULL  |
| TURNER | 1500.00 | SALES      |     3 | BLAKE |
| ADAMS  | 1100.00 | RESEARCH   |     1 | SCOTT |
| JAMES  |  950.00 | SALES      |     1 | BLAKE |
| FORD   | 3000.00 | RESEARCH   |     4 | JONES |
| MILLER | 1300.00 | ACCOUNTING |     2 | CLARK |
+--------+---------+------------+-------+-------+
```

3、子查询

3.1、什么是子查询?

select语句中嵌套select语句，被嵌套的select语句称为子查询。

3.2、子查询都可以出现在哪里呢?

```
    select
        ..(select).
    from
        ..(select).
    where
        ..(select).
```

3.3、where子句中的子查询

案例：找出比最低工资高的员工姓名和工资？

```
    select
        ename,sal
    from
        emp
    where
        sal > min(sal);

    ERROR 1111 (HY000): Invalid use of group
function
```

where子句中不能直接使用分组函数。

实现思路：

第一步：查询最低工资是多少

```
        select min(sal) from emp;
        +----------+
        | min(sal) |
        +----------+
        |   800.00 |
        +----------+
```

第二步：找出>800的

```
        select ename,sal from emp where sal > 800;
```

第三步：合并

```
        select ename,sal from emp where sal >
(select min(sal) from emp);
        +--------+---------+
```

```
| ename  | sal     |
+--------+---------+
| ALLEN  | 1600.00 |
| WARD   | 1250.00 |
| JONES  | 2975.00 |
| MARTIN | 1250.00 |
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| SCOTT  | 3000.00 |
| KING   | 5000.00 |
| TURNER | 1500.00 |
| ADAMS  | 1100.00 |
| JAMES  |  950.00 |
| FORD   | 3000.00 |
| MILLER | 1300.00 |
+--------+---------+
```

3.4、from子句中的子查询

注意：from后面的子查询，可以将子查询的查询结果当做一张临时表。（技巧）

案例：找出每个岗位的平均工资的薪资等级。

第一步：找出每个岗位的平均工资（按照岗位分组求平均值）

```
select job,avg(sal) from emp group by job;
+-----------+-------------+
| job       | avgsal      |
+-----------+-------------+
| ANALYST   | 3000.000000 |
| CLERK     | 1037.500000 |
| MANAGER   | 2758.333333 |
| PRESIDENT | 5000.000000 |
| SALESMAN  | 1400.000000 |
+-----------+-------------+t表
```

第二步：克服心理障碍，把以上的查询结果就当做一张真实存在的表t。

```
mysql> select * from salgrade; s表
+-------+-------+-------+
| GRADE | LOSAL | HISAL |
+-------+-------+-------+
|     1 |   700 |  1200 |
|     2 |  1201 |  1400 |
|     3 |  1401 |  2000 |
|     4 |  2001 |  3000 |
|     5 |  3001 |  9999 |
+-------+-------+-------+
t表和s表进行表连接，条件：t表avg(sal) between s.losal and s.hisal;

    select
        t.*, s.grade
    from
        (select job,avg(sal) as avgsal from emp group by job) t
    join
        salgrade s
    on
        t.avgsal between s.losal and s.hisal;

    +-----------+-------------+-------+
    | job       | avgsal      | grade |
    +-----------+-------------+-------+
    | CLERK     | 1037.500000 |     1 |
    | SALESMAN  | 1400.000000 |     2 |
    | ANALYST   | 3000.000000 |     4 |
    | MANAGER   | 2758.333333 |     4 |
    | PRESIDENT | 5000.000000 |     5 |
    +-----------+-------------+-------+
```

3.5、select后面出现的子查询（这个内容不需要掌握，了解即可！！！）

案例：找出每个员工的部门名称，要求显示员工名，部门名?

```
select
e.ename,e.deptno,(select d.dname from dept d where
e.deptno = d.deptno) as dname
from
emp e;
```

```
+--------+--------+------------+
| ename  | deptno | dname      |
+--------+--------+------------+
| SMITH  |     20 | RESEARCH   |
| ALLEN  |     30 | SALES      |
| WARD   |     30 | SALES      |
| JONES  |     20 | RESEARCH   |
| MARTIN |     30 | SALES      |
| BLAKE  |     30 | SALES      |
| CLARK  |     10 | ACCOUNTING |
| SCOTT  |     20 | RESEARCH   |
| KING   |     10 | ACCOUNTING |
| TURNER |     30 | SALES      |
| ADAMS  |     20 | RESEARCH   |
| JAMES  |     30 | SALES      |
| FORD   |     20 | RESEARCH   |
| MILLER |     10 | ACCOUNTING |
+--------+--------+------------+

//以下是错误的：ERROR 1242 (21000): Subquery returns
more than 1 row
select
    e.ename,e.deptno,(select dname from dept) as
dname
from
```

```
    emp e;
```

注意：对于select后面的子查询来说，这个子查询只能一次返回1条结果，
多于1条，就报错了。！

4、union合并查询结果集

案例：查询工作岗位是MANAGER和SALESMAN的员工?

```
select ename,job from emp where job = 'MANAGER' or
job = 'SALESMAN';
select ename,job from emp where job
in('MANAGER','SALESMAN');
+--------+----------+
| ename  | job      |
+--------+----------+
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
| JONES  | MANAGER  |
| MARTIN | SALESMAN |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| TURNER | SALESMAN |
+--------+----------+
```

```
select ename,job from emp where job = 'MANAGER'
union
select ename,job from emp where job = 'SALESMAN';
+--------+----------+
| ename  | job      |
+--------+----------+
| JONES  | MANAGER  |
| BLAKE  | MANAGER  |
| CLARK  | MANAGER  |
| ALLEN  | SALESMAN |
| WARD   | SALESMAN |
```

```
| MARTIN | SALESMAN |
| TURNER | SALESMAN |
+--------+----------+
```

union的效率要高一些。对于表连接来说，每连接一次新表，则匹配的次数满足笛卡尔积，成倍的翻。。。
但是union可以减少匹配的次数。在减少匹配次数的情况下，还可以完成两个结果集的拼接。

a 连接 b 连接 c
a 10条记录
b 10条记录
c 10条记录
匹配次数是：1000

a 连接 b一个结果：10 * 10 --> 100次
a 连接 c一个结果：10 * 10 --> 100次
使用union的话是：100次 + 100次 = 200次。（union把乘法变成了加法运算）

union在使用的时候有注意事项吗？

```
//以下是错误的：union在进行结果集合并的时候，要求两个结果集的列数相同。
select ename,job from emp where job = 'MANAGER'
union
select ename from emp where job = 'SALESMAN';

//以下MYSQL可以，oracle语法严格 ，不可以，报错。要求：结果集合并时列和列的数据类型也要一致。
select ename,job from emp where job = 'MANAGER'
union
select ename,sal from emp where job = 'SALESMAN';
+--------+---------+
| ename  | job     |
+--------+---------+
```

```
| JONES   | MANAGER |
| BLAKE   | MANAGER |
| CLARK   | MANAGER |
| ALLEN   | 1600    |
| WARD    | 1250    |
| MARTIN  | 1250    |
| TURNER  | 1500    |
+--------+---------+
```

★★★5、limit（非常重要）

5.1、limit作用：将查询结果集的一部分取出来。通常使用在分页查询当中。

百度默认：一页显示10条记录。
分页的作用是为了提高用户的体验，因为一次全部都查出来，用户体验差。
可以一页一页翻页看。

5.2、limit怎么用呢?

```
完整用法：limit startIndex, length
    startIndex是起始下标，length是长度。
    起始下标从0开始。

缺省用法：limit 5; 这是取前5.

按照薪资降序，取出排名在前5名的员工？
select
    ename,sal
from
    emp
order by
    sal desc
limit 5; //取前5
```

```
select
    ename,sal
from
    emp
order by
    sal desc
limit 0,5;

+-------+---------+
| ename | sal     |
+-------+---------+
| KING  | 5000.00 |
| SCOTT | 3000.00 |
| FORD  | 3000.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
+-------+---------+
```

5.3、注意：mysql当中limit在order by之后执行！！

5.4、取出工资排名在[3-5]名的员工?

```
select
ename,sal
from
emp
order by
sal desc
limit
2, 3;
```

2表示起始位置从下标2开始，就是第三条记录。
3表示长度。

```
+-------+---------+
| ename | sal     |
+-------+---------+
| FORD  | 3000.00 |
| JONES | 2975.00 |
| BLAKE | 2850.00 |
+-------+---------+
```

5.5、取出工资排名在[5-9]名的员工

```
select
ename,sal
from
emp
order by
sal desc
limit
4, 5;

+--------+---------+
| ename  | sal     |
+--------+---------+
| BLAKE  | 2850.00 |
| CLARK  | 2450.00 |
| ALLEN  | 1600.00 |
| TURNER | 1500.00 |
| MILLER | 1300.00 |
+--------+---------+
```

5.6、分页

每页显示3条记录

第1页：limit 0,3　　[0 1 2]

第2页：limit 3,3　　[3 4 5]

第3页：limit 6,3　　[6 7 8]

第4页：limit 9,3　　[9 10 11]

每页显示pageSize条记录

第pageNo页：limit (pageNo - 1) * pageSize ，pageSize

示例：

```
static void main(string[] args){
    // 用户提交过来一个页码，以及每页显示的记录条数
    int pageNo = 5; //第5页
    int pageSize = 10; //每页显示10条

    int startIndex = (pageNo - 1) * pageSize;
    String sql = "select ...limit " + startIndex +
", " + pageSize;
}
```

记公式：

```
limit (pageNo-1)*pageSize , pageSize
```

6、关于DQL语句的大总结：

```
    select
        ...
    from
        ...
    where
        ...
    group by
        ...
    having
```

```
        ...
    order by
        ...
    limit
        ...
```

执行顺序?

1.from
2.where
3.group by
4.having
5.select
6.order by
7.limit..

7、表的创建（建表）-DDL

7.1、建表的语法格式：(建表属于DDL语句，DDL包括：create drop alter)

```
create table 表名(字段名1 数据类型, 字段名2 数据类型, 字段名3 数据类型);

create table 表名(
    字段名1 数据类型,
    字段名2 数据类型,
    字段名3 数据类型
);

表名：建议以t_ 或者 tbl_开始，可读性强。见名知意。
字段名：见名知意。
表名和字段名都属于标识符。
```

7.2、关于mysql中的数据类型?

很多数据类型，我们只需要掌握一些常见的数据类型即可。

varchar(最长255)

可变长度的字符串

比较智能，节省空间。

会根据实际的数据长度动态分配空间。

优点：节省空间

缺点：需要动态分配空间，速度慢。

char(最长255)

定长字符串

不管实际的数据长度是多少。

分配固定长度的空间去存储数据。

使用不恰当的时候，可能会导致空间的浪费。

优点：不需要动态分配空间，速度快。

缺点：使用不当可能会导致空间的浪费。

varchar和char我们应该怎么选择？

性别字段你选什么？因为性别是固定长度的字符串，所以选择char。

姓名字段你选什么？每一个人的名字长度不同，所以选择varchar。

int(最长11)

数字中的整数型。等同于C++的int。

bigint

数字中的长整型。等同于C++中的long int。

float

单精度浮点型数据

double

双精度浮点型数据

date

短日期类型

datetime

长日期类型

clob

字符大对象

最多可以存储4G的字符串。

比如：存储一篇文章，存储一个说明。

超过255个字符的都要采用CLOB字符大对象来存储。

Character Large OBject:CLOB

blob

二进制大对象

Binary Large OBject

专门用来存储图片、声音、视频等流媒体数据。

往BLOB类型的字段上插入数据的时候，例如插入一个图片、视频等，

你需要使用IO流才行。

```
t_movie 电影表（专门存储电影信息的）

编号              名字              故事情节                    上映
日期              时长                        海报       类型

no(bigint)  name(varchar)   history(clob)
playtime(date)      time(double)    image(blob)
type(char)
-------------------------------------------------------
-------------------------------------------------------
----------

10000             哪吒                ...........
2019-10-11          2.5          ....         '1'
10001             林正英之娘娘    ...........
2019-11-11          1.5            ....         '2'
....
```

7.3、创建一个学生表?

学号、姓名、年龄、性别、邮箱地址

```
create table t_student(
no int,
name varchar(32),
sex char(1),
age int(3),
email varchar(255)
);
```

删除表：

```
    drop table t_student; // 当这张表不存在的时候会报错！

    // 如果这张表存在的话，删除
    drop table if exists t_student;
```

7.4、插入数据insert （DML）

语法格式：

```
    insert into 表名(字段名1,字段名2,字段名3...)
values(值1,值2,值3);

    注意：字段名和值要一一对应。什么是一一对应？
        数量要对应。数据类型要对应。

insert into t_student(no,name,sex,age,email)
values(1,'zhangsan','m',20,'zhangsan@123.com');
insert into t_student(email,name,sex,age,no)
values('lisi@123.com','lisi','f',20,2);

insert into t_student(no) values(3);

+------+----------+------+------+-----------------+
| no   | name     | sex  | age  | email           |
+------+----------+------+------+-----------------+
|    1 | zhangsan | m    |   20 | zhangsan@123.com |
|    2 | lisi     | f    |   20 | lisi@123.com     |
|    3 | NULL     | NULL | NULL | NULL            |
+------+----------+------+------+-----------------+
insert into t_student(name) values('wangwu');
+------+----------+------+------+-----------------+
| no   | name     | sex  | age  | email           |
+------+----------+------+------+-----------------+
|    1 | zhangsan | m    |   20 | zhangsan@123.com |
|    2 | lisi     | f    |   20 | lisi@123.com     |
|    3 | NULL     | NULL | NULL | NULL            |
```

```
| NULL | wangwu   | NULL | NULL | NULL                |
+------+----------+------+------+---------------------+
```

注意：insert语句但凡是执行成功了，那么必然会多一条记录。

没有给其它字段指定值的话，默认值是NULL。

```
drop table if exists t_student;
create table t_student(
    no int,
    name varchar(32),
    sex char(1) default 'm',
    age int(3),
    email varchar(255)
);

+-------+-------------+------+-----+---------+-----
--+
| Field | Type        | Null | Key | Default |
Extra |
+-------+-------------+------+-----+---------+-----
--+
| no    | int(11)     | YES  |     | NULL    |
  |
| name  | varchar(32) | YES  |     | NULL    |
  |
| sex   | char(1)     | YES  |     | m       |
  |
| age   | int(3)      | YES  |     | NULL    |
  |
| email | varchar(255) | YES  |     | NULL    |
  |
+-------+-------------+------+-----+---------+-----
--+
insert into t_student(no) values(1);
mysql> select * from t_student;
+------+------+------+------+-------+
| no   | name | sex  | age  | email |
+------+------+------+------+-------+
```

```
|    1 | NULL | m    | NULL | NULL  |
+------+------+------+------+-------+

insert语句中的“字段名”可以省略吗？可以
    insert into t_student values(2); //错误的

    // 注意：前面的字段名省略的话，等于都写上了！所以值也要都写上！
    insert into t_student values(2, 'lisi', 'f', 20, 'lisi@123.com');
    +------+------+------+------+--------------+
    | no   | name | sex  | age  | email        |
    +------+------+------+------+--------------+
    |    1 | NULL | m    | NULL | NULL         |
    |    2 | lisi | f    |   20 | lisi@123.com |
    +------+------+------+------+--------------+
```

7.5、insert插入日期

```
数字格式化：format
    select ename,sal from emp;
    +--------+---------+
    | ename  | sal     |
    +--------+---------+
    | SMITH  |  800.00 |
    | ALLEN  | 1600.00 |
    | WARD   | 1250.00 |
    | JONES  | 2975.00 |
    | MARTIN | 1250.00 |
    | BLAKE  | 2850.00 |
    | CLARK  | 2450.00 |
    | SCOTT  | 3000.00 |
    | KING   | 5000.00 |
    | TURNER | 1500.00 |
    | ADAMS  | 1100.00 |
    | JAMES  |  950.00 |
```

```
    | FORD   | 3000.00 |
    | MILLER | 1300.00 |
    +--------+---------+
```

格式化数字：format(数字, '格式')

```
    select ename,format(sal, '$999,999') as sal
from emp;
```

```
    +--------+-------+
    | ename  | sal   |
    +--------+-------+
    | SMITH  | 800   |
    | ALLEN  | 1,600 |
    | WARD   | 1,250 |
    | JONES  | 2,975 |
    | MARTIN | 1,250 |
    | BLAKE  | 2,850 |
    | CLARK  | 2,450 |
    | SCOTT  | 3,000 |
    | KING   | 5,000 |
    | TURNER | 1,500 |
    | ADAMS  | 1,100 |
    | JAMES  | 950   |
    | FORD   | 3,000 |
    | MILLER | 1,300 |
    +--------+-------+
```

str_to_date：将字符串varchar类型转换成date类型
date_format：将date类型转换成具有一定格式的varchar字符串类型。

```
drop table if exists t_user;
create table t_user(
    id int,
    name varchar(32),
    birth date // 生日也可以使用date日期类型
);
```

```
create table t_user(
    id int,
    name varchar(32),
    birth char(10) // 生日可以使用字符串，没问题。
);
```

生日：1990-10-11 （10个字符）

注意：数据库中的有一条命名规范：
    所有的标识符都是全部小写，单词和单词之间使用下划线进行衔接。

```
mysql> desc t_user;
+-------+------------+------+-----+---------+------
-+
| Field | Type       | Null | Key | Default | Extra
|
+-------+------------+------+-----+---------+------
-+
| id    | int(11)    | YES  |     | NULL    |
|
| name  | varchar(32) | YES  |     | NULL    |
|
| birth | date       | YES  |     | NULL    |
|
+-------+------------+------+-----+---------+------
-+
```

插入数据？
    `insert into t_user(id,name,birth) values(1, 'zhangsan', '01-10-1990'); // 1990年10月1日`
    出问题了：原因是类型不匹配。数据库birth是date类型，这里给了一个字符串varchar。

    怎么办？可以使用str_to_date函数进行类型转换。
    str_to_date函数可以将字符串转换成日期类型date
    语法格式：

```
str_to_date('字符串日期', '日期格式')
```

mysql的日期格式：

```
%Y  年
%m 月
%d 日
%h  时
%i  分
%s  秒
```

```
insert into t_user(id,name,birth) values(1, 'zhangsan', str_to_date('01-10-1990','%d-%m-%Y'));
```

str_to_date函数可以把字符串varchar转换成日期date类型数据，

通常使用在插入insert方面，因为插入的时候需要一个日期类型的数据，

需要通过该函数将字符串转换成date。

好消息

如果你提供的日期字符串是这个格式，str_to_date函数就不需要了！！！

```
%Y-%m-%d
insert into t_user(id,name,birth) values(2, 'lisi', '1990-10-01');
```

查询的时候可以以某个特定的日期格式展示吗？

date_format

这个函数可以将日期类型转换成特定格式的字符串。

```
select id,name,date_format(birth, '%m/%d/%Y') as birth from t_user;
```

```
+------+----------+------------+
| id   | name     | birth      |
+------+----------+------------+
|    1 | zhangsan | 10/01/1990 |
|    2 | lisi     | 10/01/1990 |
```

```
    +------+----------+------------+

    date_format函数怎么用？
        date_format(日期类型数据, '日期格式')
        这个函数通常使用在查询日期方面。设置展示的日期格式。

    mysql> select id,name,birth from t_user;
    +------+----------+------------+
    | id   | name     | birth      |
    +------+----------+------------+
    |    1 | zhangsan | 1990-10-01 |
    |    2 | lisi     | 1990-10-01 |
    +------+----------+------------+
    以上的SQL语句实际上是进行了默认的日期格式化，
    自动将数据库中的date类型转换成varchar类型。
    并且采用的格式是mysql默认的日期格式：'%Y-%m-%d'

    select id,name,date_format(birth,'%Y/%m/%d') as
birth from t_user;

    java中的日期格式？
        yyyy-MM-dd HH:mm:ss SSS
```

7.6、date和datetime两个类型的区别

date是短日期：只包括年月日信息。
datetime是长日期：包括年月日时分秒信息。

```
drop table if exists t_user;
create table t_user(
    id int,
    name varchar(32),
    birth date,
    create_time datetime
);
```

```
id是整数
name是字符串
birth是短日期
create_time是这条记录的创建时间：长日期类型

mysql短日期默认格式：%Y-%m-%d
mysql长日期默认格式：%Y-%m-%d %h:%i:%s

insert into t_user(id,name,birth,create_time)
values(1,'zhangsan','1990-10-01','2020-03-18
15:49:50');

在mysql当中怎么获取系统当前时间？
    now() 函数，并且获取的时间带有：时分秒信息！！！！是
datetime类型的。

    insert into t_user(id,name,birth,create_time)
values(2,'lisi','1991-10-01',now());
```

7.7、修改update（DML）

语法格式：

```
update 表名 set 字段名1=值1,字段名2=值2,字段名3=值3...
where 条件;
```

注意：没有条件限制会导致所有数据全部更新。

```
update t_user set name = 'jack', birth = '2000-10-
11' where id = 2;
+------+----------+------------+--------------------
-+
| id   | name     | birth      | create_time
|
+------+----------+------------+--------------------
-+
|    1 | zhangsan | 1990-10-01 | 2020-03-18 15:49:50
|
|    2 | jack     | 2000-10-11 | 2020-03-18 15:51:23
|
+------+----------+------------+--------------------
-+
```

```
update t_user set name = 'jack', birth = '2000-10-
11', create_time = now() where id = 2;
```

更新所有?

```
update t_user set name = 'abc';
```

7.8、删除数据 delete （DML）

语法格式?

```
delete from 表名 where 条件;
```

```
注意：没有条件，整张表的数据会全部删除！

delete from t_user where id = 2;

insert into t_user(id) values(2);

delete from t_user; // 删除所有！
```

day03

1、查询每一个员工的所在部门名称？要求显示员工名和部门名。

```
mysql> select * from emp;
+-------+--------+-----------+------+------------+---
-------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   |
SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+---
-------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |
 800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 |
1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 |
1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 |
2975.00 |    NULL |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 |
1250.00 | 1400.00 |     30 |
```

```
|  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 |
2850.00 |    NULL |     30 |
|  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 |
2450.00 |    NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 |
3000.00 |    NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 |
5000.00 |    NULL |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 |
1500.00 |    0.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 |
1100.00 |    NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |
 950.00 |    NULL |     30 |a
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 |
3000.00 |    NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 1982-01-23 |
1300.00 |    NULL |     10 |
+-------+--------+-----------+------+------------+--
-------+---------+--------+
```

```
mysql> select * from dept;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

从emp表中取ename，从dept表中取dname，没有条件限制最终查询结果是?

```
ENAME                DNAME

SMITH                ACCOUNTING   无效记录
```

```
SMITH                RESEARCH          有效记录
SMITH                SALES             无效记录
SMITH                OPERATIONS    无效记录

ALLEN                ACCOUNTING
ALLEN                RESEARCH
ALLEN                SALES
ALLEN                OPERATIONS


.....
56条记录。
```

加个条件是为了达到4选1，也是为了数据的有效性。

```
select
    e.ename,d.dname
from
    emp e
join
    dept d
on
    e.deptno = d.deptno;
```

加条件只是为了避免笛卡尔积现象，只是为了查询出有效的组合记录。
匹配的次数一次都没有少，还是56次。

2、insert语句可以一次插入多条记录

可以的!

```
mysql> desc t_user;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | int(11)     | YES  |     | NULL    |       |
| name        | varchar(32) | YES  |     | NULL    |       |
| birth       | date        | YES  |     | NULL    |       |
| create_time | datetime    | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

一次可以插入多条记录：

```
insert into t_user(id,name,birth,create_time) values
(1,'zs','1980-10-11',now()),
(2,'lisi','1981-10-11',now()),
(3,'wangwu','1982-10-11',now());
```

语法：insert into t_user(字段名1,字段名2) values(),(),(),();

```
mysql> select * from t_user;
+------+--------+------------+---------------------+
| id   | name   | birth      | create_time         |
+------+--------+------------+---------------------+
|    1 | zs     | 1980-10-11 | 2020-03-19 09:37:01 |
|    2 | lisi   | 1981-10-11 | 2020-03-19 09:37:01 |
|    3 | wangwu | 1982-10-11 | 2020-03-19 09:37:01 |
+------+--------+------------+---------------------+
```

3、快速创建表【了解内容】

```
mysql> create table emp2 as select * from emp;

原理：
    将一个查询结果当做一张表新建！！！！！
    这个可以完成表的快速复制！！！！
    表创建出来，同时表中的数据也存在了！！！

create table mytable as select empno,ename from emp
where job = 'MANAGER';
```

4、将查询结果插入到一张表当中？insert相关的！！！【了解内容】

```
create table dept_bak as select * from dept;
mysql> select * from dept_bak;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

```
insert into dept_bak select * from dept; //很少用！

mysql> select * from dept_bak;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
```

```
|      40 | OPERATIONS | BOSTON   |
|      10 | ACCOUNTING | NEW YORK |
|      20 | RESEARCH   | DALLAS   |
|      30 | SALES      | CHICAGO  |
|      40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

5、快速删除表中的数据【truncate比较重要，必须掌握】

```
//删除dept_bak表中的数据
delete from dept_bak; //这种删除数据的方式比较慢。

mysql> select * from dept_bak;
Empty set (0.00 sec)
```

delete语句删除数据的原理？（delete属于DML语句！！！）

表中的数据被删除了，但是这个数据在硬盘上的真实存储空间不会被释放！！！

这种删除缺点是：删除效率比较低。

这种删除优点是：支持回滚，后悔了可以再恢复数据！！！

truncate语句删除数据的原理？

这种删除效率比较高，表被一次截断，物理删除。

这种删除缺点：不支持回滚。

这种删除优点：快速。

用法：truncate table dept_bak; （这种操作属于DDL操作。）

大表非常大，上亿条记录？？？？

删除的时候，使用delete，也许需要执行1个小时才能删除完！效率较低。

可以选择使用truncate删除表中的数据。只需要不到1秒钟的时间就删除结束。效率较高。

但是使用truncate之前，必须仔细询问客户是否真的要删除，并警告删除之后不可恢复！

```
    truncate是删除表中的数据，表还在！

删除表操作？
    drop table 表名; // 这不是删除表中的数据，这是把表删除。
```

6、对表结构的增删改

什么是对表结构的修改？
　添加一个字段，删除一个字段，修改一个字段！！！

对表结构的修改需要使用：alter
属于DDL语句

DDL包括：create drop alter

第一：在实际的开发中，需求一旦确定之后，表一旦设计好之后，很少的
进行表结构的修改。因为开发进行中的时候，修改表结构，成本比较高。
修改表的结构，对应的代码就需要进行大量的修改。成本是比较高的。
这个责任应该由设计人员来承担！

第二：由于修改表结构的操作很少，所以我们不需要掌握，如果有一天
真的要修改表结构，你可以使用工具！！！！

修改表结构的操作是不需要写到程序中的。实际上也不是程序员的范畴。

★★★7、约束

7.1、什么是约束?

约束对应的英语单词：constraint

在创建表的时候，我们可以给表中的字段加上一些约束，来保证这个表中数据的

完整性、有效性！！！

约束的作用就是为了保证：表中的数据有效！！

7.2、约束包括哪些?

非空约束：not null

唯一性约束: unique

主键约束: primary key （简称PK)

外键约束：foreign key（简称FK)

检查约束：check（mysql不支持，oracle支持)

```
我们这里重点学习四个约束：
非空约束      not null
唯一性约束    unique
主键约束      primary key
外键约束      foreign key
```

7.3、非空约束：not null

```
非空约束not null约束的字段不能为NULL。
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255) not null  // not null只有列级约束，没有表级约束！
);
insert into t_vip(id,name) values(1,'zhangsan');
insert into t_vip(id,name) values(2,'lisi');
```

```
insert into t_vip(id) values(3);
ERROR 1364 (HY000): Field 'name' doesn't have a
default value

小插曲：
    xxxx.sql这种文件被称为sql脚本文件。
    sql脚本文件中编写了大量的sql语句。
    我们执行sql脚本文件的时候，该文件中所有的sql语句会全部执
行！
    批量的执行SQL语句，可以使用sql脚本文件。
    在mysql当中怎么执行sql脚本呢？
        mysql> source D:\course\03-
MySQL\document\vip.sql

    你在实际的工作中，第一天到了公司，项目经理会给你一个
xxx.sql文件，
    你执行这个脚本文件，你电脑上的数据库数据就有了！
```

7.4、唯一性约束: unique

唯一性约束unique约束的字段不能重复，但是可以为NULL。

```
drop table if exists t_vip;
create table t_vip(
    id int,
    name varchar(255) unique,
    email varchar(255)
);
insert into t_vip(id,name,email)
values(1,'zhangsan','zhangsan@123.com');
insert into t_vip(id,name,email)
values(2,'lisi','lisi@123.com');
insert into t_vip(id,name,email)
values(3,'wangwu','wangwu@123.com');
select * from t_vip;
```

```
insert into t_vip(id,name,email)
values(4,'wangwu','wangwu@sina.com');
ERROR 1062 (23000): Duplicate entry 'wangwu' for key
'name'

insert into t_vip(id) values(4);
insert into t_vip(id) values(5);
+------+----------+------------------+
| id   | name     | email            |
+------+----------+------------------+
|    1 | zhangsan | zhangsan@123.com |
|    2 | lisi     | lisi@123.com     |
|    3 | wangwu   | wangwu@123.com   |
|    4 | NULL     | NULL             |
|    5 | NULL     | NULL             |
+------+----------+------------------+
```

name字段虽然被unique约束了，但是可以为NULL。

新需求：name和email两个字段联合起来具有唯一性！！！！

列级约束

```
    drop table if exists t_vip;
    create table t_vip(
        id int,
        name varchar(255) unique,  // 约束直接添加到列
后面的，叫做列级约束。
        email varchar(255) unique
    );
```

这张表这样创建是不符合我以上“新需求”的。

这样创建表示：name具有唯一性，email具有唯一性。各自唯一。

以下这样的数据是符合我“新需求”的。

但如果采用以上方式创建表的话，肯定创建失败，因为'zhangsan'和'zhangsan'重复了。

```
insert into t_vip(id,name,email)
values(1,'zhangsan','zhangsan@123.com');
insert into t_vip(id,name,email)
values(2,'zhangsan','zhangsan@sina.com');
```

怎么创建这样的表，才能符合新需求呢?

```
drop table if exists t_vip;
create table t_vip(
   id int,
   name varchar(255),
   email varchar(255),
   unique(name,email) // 约束没有添加在列的后面，这种约束被称为表级约束。
);
insert into t_vip(id,name,email)
values(1,'zhangsan','zhangsan@123.com');
insert into t_vip(id,name,email)
values(2,'zhangsan','zhangsan@sina.com');
select * from t_vip;

name和email两个字段联合起来唯一！！！
insert into t_vip(id,name,email)
values(3,'zhangsan','zhangsan@sina.com');
ERROR 1062 (23000): Duplicate entry 'zhangsan-zhangsan@sina.com' for key 'name'
```

什么时候使用**表级约束**呢?

需要给多个字段联合起来添加某一个约束的时候，需要使用表级约束。

```
unique 和not null可以联合吗？
    drop table if exists t_vip;
    create table t_vip(
        id int,
        name varchar(255) not null unique
    );
    mysql> desc t_vip;
    +-------+--------------+------+-----+---------+-
------+
    | Field | Type         | Null | Key | Default |
Extra |
    +-------+--------------+------+-----+---------+-
------+
    | id    | int(11)      | YES  |     | NULL    |
      |
    | name  | varchar(255) | NO   | PRI | NULL    |
      |
    +-------+--------------+------+-----+---------+-
------+
```

在mysql当中，如果一个字段同时被not null和unique约束的话，该字段自动变成主键字段。（注意：oracle中不一样！）

```
insert into t_vip(id,name) values(1,'zhangsan');
insert into t_vip(id,name) values(2,'zhangsan'); //
错误了：name不能重复
insert into t_vip(id) values(2); //错误了：name不能为
NULL。
```

★★★7.5、主键约束（primary key，简称PK）非常重要五颗星*

主键约束的相关术语？

主键约束：就是一种约束。

主键字段：该字段上添加了主键约束，这样的字段叫做：主键字段

主键值：主键字段中的每一个值都叫做：主键值。

什么是主键？有啥用？

主键值是每一行记录的唯一标识。

主键值是每一行记录的身份证号！！！

记住：任何一张表都应该有主键，没有主键，表无效！！

主键的特征：not null + unique（主键值不能是NULL，同时也不能重复！）

怎么给一张表添加主键约束呢？

```
    drop table if exists t_vip;
    // 1个字段做主键，叫做：单一主键
    create table t_vip(
        id int primary key,  //列级约束
        name varchar(255)
    );
    insert into t_vip(id,name) values(1,'zhangsan');
    insert into t_vip(id,name) values(2,'lisi');

    //错误：不能重复
    insert into t_vip(id,name) values(2,'wangwu');
    ERROR 1062 (23000): Duplicate entry '2' for key
'PRIMARY'

    //错误：不能为NULL
    insert into t_vip(name) values('zhaoliu');
```

```
    ERROR 1364 (HY000): Field 'id' doesn't have a
default value
```

可以这样添加主键吗，使用表级约束？

```
drop table if exists t_vip;
    create table t_vip(
        id int,
        name varchar(255),
        primary key(id)  // 表级约束
    );
    insert into t_vip(id,name) values(1,'zhangsan');

    //错误
    insert into t_vip(id,name) values(1,'lisi');
    ERROR 1062 (23000): Duplicate entry '1' for key
'PRIMARY'
```

表级约束主要是给多个字段联合起来添加约束？

```
drop table if exists t_vip;
    // id和name联合起来做主键：复合主键！！！！
    create table t_vip(
        id int,
        name varchar(255),
        email varchar(255),
        primary key(id,name)
    );
    insert into t_vip(id,name,email)
values(1,'zhangsan','zhangsan@123.com');
    insert into t_vip(id,name,email)
values(1,'lisi','lisi@123.com');

    //错误：不能重复
    insert into t_vip(id,name,email)
values(1,'lisi','lisi@123.com');
```

```
    ERROR 1062 (23000): Duplicate entry '1-lisi' for
key 'PRIMARY'
```

在实际开发中不建议使用：复合主键。建议使用单一主键！

因为主键值存在的意义就是这行记录的身份证号，只要意义达到即可，单一主键可以做到。

复合主键比较复杂，不建议使用！！！

一个表中主键约束能加两个吗？

```
drop table if exists t_vip;
    create table t_vip(
        id int primary key,
        name varchar(255) primary key
    );
    ERROR 1068 (42000): Multiple primary key defined
```

结论：一张表，主键约束只能添加1个。（主键只能有1个。）

主键值建议使用：

int

bigint

char

等类型。

不建议使用：varchar来做主键。主键值一般都是数字，一般都是定长的！

主键除了：单一主键和复合主键之外，还可以这样进行分类？

自然主键：主键值是一个自然数，和业务没关系。

业务主键：主键值和业务紧密关联，例如拿银行卡账号做主键值。这就是业务主键！

在实际开发中使用业务主键多，还是使用自然主键多一些？

自然主键使用比较多，因为主键只要做到不重复就行，不需要有意义。

业务主键不好，因为主键一旦和业务挂钩，那么当业务发生变动的时候，

可能会影响到主键值，所以业务主键不建议使用。尽量使用自然主键。

在mysql当中，有一种机制，可以帮助我们自动维护一个主键值？

auto_increment

```
    drop table if exists t_vip;
    create table t_vip(
        id int primary key auto_increment,
//auto_increment表示自增，从1开始，以1递增!
        name varchar(255)
    );
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    insert into t_vip(name) values('zhangsan');
    select * from t_vip;

    +----+----------+
    | id | name     |
    +----+----------+
    |  1 | zhangsan |
    |  2 | zhangsan |
    |  3 | zhangsan |
    |  4 | zhangsan |
    |  5 | zhangsan |
    |  6 | zhangsan |
```

```
|  7 | zhangsan |
|  8 | zhangsan |
+----+----------+
```

★★★7.6、外键约束（foreign key，简称FK）非常重要五颗星*

外键约束涉及到的相关术语：

外键约束：一种约束（foreign key）

外键字段：该字段上添加了外键约束

外键值：外键字段当中的每一个值。

```
业务背景：
    请设计数据库表，来描述“班级和学生”的信息？

    第一种方案：班级和学生存储在一张表中？？？
    t_student
    no(pk)          name        classno         classname
    ------------------------------------------------------------------
    1               jack        100             北京市大兴区亦庄镇第二中学高三1班
    2               lucy        100             北京市大兴区亦庄镇第二中学高三1班
    3               lilei       100             北京市大兴区亦庄镇第二中学高三1班
    4               hanmeimei   100             北京市大兴区亦庄镇第二中学高三1班
    5               zhangsan    101             北京市大兴区亦庄镇第二中学高三2班
    6               lisi        101             北京市大兴区亦庄镇第二中学高三2班
```

7 wangwu 101 北京市大兴区亦庄镇第二中学高三2班
8 zhaoliu 101 北京市大兴区亦庄镇第二中学高三2班

分析以上方案的缺点：

数据冗余，空间浪费！！！！

这个设计是比较失败的！

第二种方案：班级一张表、学生一张表？？

t_class 班级表

classno(pk)	classname
100	北京市大兴区亦庄镇第二中学高三1班
101	北京市大兴区亦庄镇第二中学高三1班

t_student 学生表

no(pk)	name	cno(FK引用t_class这张表的classno)
1	jack	100
2	lucy	100
3	lilei	100
4	hanmeimei	100
5	zhangsan	101
6	lisi	101
7	wangwu	101
8	zhaoliu	101

当cno字段没有任何约束的时候，可能会导致数据无效。可能出现一个102，但是102班级不存在。

所以为了保证cno字段中的值都是100和101，需要给cno字段添加外键约束。

那么：cno字段就是外键字段。cno字段中的每一个值都是外键值。

注意：

t_class是父表

t_student是子表

删除表的顺序？

先删子，再删父。

创建表的顺序？

先创建父，再创建子。

删除数据的顺序？

先删子，再删父。

插入数据的顺序？

先插入父，再插入子。

思考：子表中的外键引用的父表中的某个字段，被引用的这个字段必须是主键吗？

不一定是主键，但至少具有unique约束。

测试：外键可以为NULL吗？

外键值可以为NULL。

```
drop table if exists t_student;
drop table if exists t_class;
create table t_class(
    classno int primary key,
    classname varchar(255)
);
create table t_student(
    no int primary key auto_increment,
    name varchar(255),
    cno int,
    foreign key(cno) references t_class(classno)
);
```

```
insert into t_class(classno,classname)
values(100,'杭州市杭州电子科技大学');
insert into t_class(classno,classname)
values(101,'浙江大学');

insert into t_student(name,cno) values('jack',100);
insert into t_student(name,cno) values('lucy',100);
insert into t_student(name,cno)
values('zhangsan',100);
insert into t_student(name,cno) values('lisi',101);

select * from t_class;
select * from t_student;
```

8、存储引擎（了解内容）

8.1、什么是存储引擎，有什么用呢?

存储引擎是MySQL中特有的一个术语，其它数据库中没有。(Oracle中有，但是不叫这个名字)

存储引擎这个名字高端大气上档次。

实际上存储引擎是一个表存储/组织数据的方式。

不同的存储引擎，表存储数据的方式不同。

8.2、怎么给表添加/指定“存储引擎”呢?

show create table t_student;

```
可以在建表的时候给表指定存储引擎。
CREATE TABLE `t_student` (
  `no` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `cno` int(11) DEFAULT NULL,
  PRIMARY KEY (`no`),
```

```
  KEY `cno` (`cno`),
  CONSTRAINT `t_student_ibfk_1` FOREIGN KEY (`cno`)
REFERENCES `t_class` (`classno`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT
CHARSET=utf8
```

在建表的时候可以在最后小括号的")"的右边使用：

ENGINE来指定存储引擎。

CHARSET来指定这张表的字符编码方式。

结论：

mysql默认的存储引擎是：InnoDB

mysql默认的字符编码方式是：utf8

建表时指定存储引擎，以及字符编码方式。

```
create table t_product(
    id int primary key,
    name varchar(255)
)engine=InnoDB default charset=gbk;
```

8.3、怎么查看mysql支持哪些存储引擎呢?

mysql> select version();

+-----------+

| version() |

+-----------+

| 5.5.36 |

+-----------+

命令： show engines \G

********** 1. row **********

Engine: FEDERATED

Support: NO

Comment: Federated MySQL storage engine

Transactions: NULL

```
        XA: NULL
  Savepoints: NULL
********* 2. row *********
     Engine: MRG_MYISAM
    Support: YES
    Comment: Collection of identical MyISAM tables
Transactions: NO
        XA: NO
  Savepoints: NO
********* 3. row *********
     Engine: MyISAM
    Support: YES
    Comment: MyISAM storage engine
Transactions: NO
        XA: NO
  Savepoints: NO
********* 4. row *********
     Engine: BLACKHOLE
    Support: YES
    Comment: /dev/null storage engine (anything you write to it
disappears
Transactions: NO
        XA: NO
  Savepoints: NO
********* 5. row *********
     Engine: CSV
    Support: YES
    Comment: CSV storage engine
Transactions: NO
        XA: NO
  Savepoints: NO
********* 6. row *********
```

```
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for
temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
********** 7. row **********
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
********** 8. row **********
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and
foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
********** 9. row **********
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
```

mysql支持九大存储引擎，当前5.5.36支持8个。版本不同支持情况不同。

8.4、关于mysql常用的存储引擎介绍一下

MyISAM存储引擎?

它管理的表具有以下特征：

使用**三个文件**表示每个表：

格式文件 — 存储表结构的定义（mytable.frm）

数据文件 — 存储表行的内容（mytable.MYD）

索引文件 — 存储表上索引（mytable.MYI）：索引是一本书的目录，缩小扫描范围，提高查询效率的一种机制。

可被转换为压缩、只读表来节省空间

提示一下：

对于一张表来说，只要是主键，

或者加有unique约束的字段上会自动创建索引。

MyISAM存储引擎特点：

可被转换为压缩、只读表来节省空间

这是这种存储引擎的优势！！！！

MyISAM不支持事务机制，安全性低。

InnoDB存储引擎?

这是mysql默认的存储引擎，同时也是一个重量级的存储引擎。

InnoDB支持事务，支持数据库崩溃后自动恢复机制。

InnoDB存储引擎最主要的特点是：非常安全。

它管理的表具有下列主要特征：

– 每个 InnoDB 表在数据库目录中以.frm 格式文件表示

– InnoDB **表空间 tablespace 被用于存储表的内容**（表空间是一个逻辑名称。表空间存储数据+索引。）

– 提供一组用来记录事务性活动的日志文件
– 用 COMMIT(提交)、SAVEPOINT 及ROLLBACK(回滚)支持事务处理
– 提供全 ACID 兼容
– 在 MySQL 服务器崩溃后提供自动恢复
– 多版本（MVCC）和行级锁定
– 支持外键及引用的完整性，包括级联删除和更新

InnoDB最大的特点就是支持事务：
以保证数据的安全。效率不是很高，并且也不能压缩，不能转换为只读，
不能很好的节省存储空间。

MEMORY存储引擎?

使用 MEMORY 存储引擎的表，其数据存储在内存中，且行的长度固定，
这两个特点使得 MEMORY 存储引擎非常快。

```
MEMORY 存储引擎管理的表具有下列特征：
    - 在数据库目录内，每个表均以.frm 格式的文件表示。
    - 表数据及索引被存储在内存中。（目的就是快，查询快！）
    - 表级锁机制。
    - 不能包含 TEXT 或 BLOB 字段。

MEMORY 存储引擎以前被称为HEAP 引擎。

MEMORY引擎优点：查询效率是最高的。不需要和硬盘交互。
MEMORY引擎缺点：不安全，关机之后数据消失。因为数据和索引都是在内存当中。
```

9、事务（重点：五颗星*，必须理解，必须掌握）

9.1、什么是事务?

一个事务其实就是一个完整的业务逻辑。
是一个最小的工作单元。不可再分。

什么是一个完整的业务逻辑?
　假设转账，从A账户向B账户中转账10000.
　将A账户的钱减去10000（update语句）
　将B账户的钱加上10000（update语句）
　这就是一个完整的业务逻辑。

　以上的操作是一个最小的工作单元，要么同时成功，要么同时失败，不可再分。
　这两个update语句要求必须同时成功或者同时失败，这样才能保证钱是正确的。

9.2、只有DML语句才会有事务这一说，其它语句和事务无关！！！

　insert
　delete
　update
　只有以上的三个语句和事务有关系，其它都没有关系。

因为 只有以上的三个语句是数据库表中数据进行增、删、改的。
只要你的操作一旦涉及到数据的增、删、改，那么就一定要考虑安全问题。

数据安全第一位！！！

9.3、假设所有的业务，只要一条DML语句就能完成，还有必要存在事务机制吗?

正是因为做某件事的时候，需要多条DML语句共同联合起来才能完成，

所以需要事务的存在。如果任何一件复杂的事儿都能一条DML语句搞定，

那么事务则没有存在的价值了。

到底什么是事务呢?

说到底，说到本质上，一个事务其实就是多条DML语句同时成功，或者同时失败!

事务：就是批量的DML语句同时成功，或者同时失败!

9.4、事务是怎么做到多条DML语句同时成功和同时失败的呢?

InnoDB存储引擎：提供一组用来记录事务性活动的日志文件

事务开启了:
insert
insert
insert
delete
update
update
update
事务结束了!

在事务的执行过程中，每一条DML的操作都会记录到"事务性活动的日志文件"中。
在事务的执行过程中，我们可以提交事务，也可以回滚事务。

提交事务?
　清空事务性活动的日志文件，将数据全部彻底持久化到数据库表中。
　提交事务标志着，事务的结束。并且是一种全部成功的结束。

回滚事务?
　将之前所有的DML操作全部撤销，并且清空事务性活动的日志文件
　回滚事务标志着，事务的结束。并且是一种全部失败的结束。

9.5、怎么提交事务，怎么回滚事务?

　提交事务：commit; 语句
　回滚事务：rollback; 语句（回滚永远都是只能回滚到上一次的提交点！）

事务对应的英语单词是：transaction

测试一下，在mysql当中默认的事务行为是怎样的?
　mysql默认情况下是支持自动提交事务的。（自动提交）
　什么是自动提交?
　　每执行一条DML语句，则提交一次！

　这种自动提交实际上是不符合我们的开发习惯，因为一个业务通常是需要多条DML语句共同执行才能完成的，为了保证数据的安全，必须要求同时成功之后再提交，所以不能执行一条就提交一条。

怎么将mysql的自动提交机制关闭掉呢?
　先执行这个命令：**start transaction;**

演示事务：

```
    ---------------------------------回滚事务---------
------------------------------
mysql> use bjpowernode;
```

```
Database changed
mysql> select * from dept_bak;
Empty set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into dept_bak values(10,'abc', 'tj');
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_bak values(10,'abc', 'tj');
Query OK, 1 row affected (0.00 sec)

mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | tj   |
|     10 | abc   | tj   |
+--------+-------+------+
2 rows in set (0.00 sec)
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from dept_bak;
Empty set (0.00 sec)
```

```
    ---------------------------------提交事务---------
------------------------------
    mysql> use bjpowernode;
    Database changed
    mysql> select * from dept_bak;
    +--------+-------+------+
    | DEPTNO | DNAME | LOC  |
    +--------+-------+------+
    |     10 | abc   | bj   |
    +--------+-------+------+
```

```
1 row in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

mysql> insert into dept_bak values(20,'abc
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
|     20 | abc   | tj   |
+--------+-------+------+
4 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from dept_bak;
+--------+-------+------+
| DEPTNO | DNAME | LOC  |
+--------+-------+------+
|     10 | abc   | bj   |
|     20 | abc   | tj   |
```

```
|      20 | abc    | tj    |
|      20 | abc    | tj    |
+--------+-------+------+
4 rows in set (0.00 sec)
```

9.6、事务包括4个特性?

A：原子性

说明事务是最小的工作单元。不可再分。

C：一致性

所有事务要求，在同一个事务当中，所有操作必须同时成功，或者同时失败，

以保证数据的一致性。

I：隔离性

A事务和B事务之间具有一定的隔离。

教室A和教室B之间有一道墙，这道墙就是隔离性。

A事务在操作一张表的时候，另一个事务B也操作这张表会那样？？？

D：持久性

事务最终结束的一个保障。事务提交，就相当于将没有保存到硬盘上的数据

保存到硬盘上！

9.7、重点研究一下事务的隔离性！！！

A教室和B教室中间有一道墙，这道墙可以很厚，也可以很薄。这就是事务的隔离级别。

这道墙越厚，表示隔离级别就越高。

事务和事务之间的隔离级别有哪些呢？4个级别

读未提交：read uncommitted（最低的隔离级别）《没有提交就读到了》

什么是读未提交?

事务A可以读取到事务B未提交的数据。

这种隔离级别存在的问题就是：

脏读现象！(Dirty Read)

我们称读到了脏数据。

这种隔离级别一般都是理论上的，大多数的数据库隔离级别都是二档起步！

读已提交：read committed《提交之后才能读到》

什么是读已提交?

事务A只能读取到事务B提交之后的数据。

这种隔离级别解决了什么问题?

解决了脏读的现象。

这种隔离级别存在什么问题?

不可重复读取数据。

什么是不可重复读取数据呢?

在事务开启之后，第一次读到的数据是3条，当前事务还没有

结束，可能第二次再读取的时候，读到的数据是4条，3不等于4

称为不可重复读取。

这种隔离级别是比较真实的数据，每一次读到的数据是绝对的真实。

oracle数据库默认的隔离级别是：read committed

可重复读：repeatable read《提交之后也读不到，永远读取的都是刚开启事务时的数据》

什么是可重复读取?

事务A开启之后，不管是多久，每一次在事务A中读取到的数据

都是一致的。即使事务B将数据已经修改，并且提交了，事

务A

读取到的数据还是没有发生改变，这就是可重复读。

可重复读解决了什么问题？

解决了不可重复读取数据。

可重复读存在的问题是什么？

可以会出现幻影读。

每一次读取到的数据都是幻象。不够真实！

早晨9点开始开启了事务，只要事务不结束，到晚上9点，读到的数据还是那样！

读到的是假象。不够绝对的真实。

mysql中默认的事务隔离级别就是这个！！！！！！！！！！！

序列化/串行化读：serializable（最高的隔离级别）

这是最高隔离级别，效率最低。解决了所有的问题。

这种隔离级别表示事务排队，不能并发！

synchronized，线程同步（事务同步）

每一次读取到的数据都是最真实的，并且效率是最低的。

9.8、验证各种隔离级别

查看隔离级别：SELECT @@transaction_isolation

```
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
```

mysql默认的隔离级别

被测试的表t_user

验证：read uncommited

mysql> set global transaction isolation level read uncommitted;

事务A	事务B
use bjpowernode;	
	use bjpowernode;
start transaction;	
select * from t_user;	
	start transaction;
	insert into t_user values('zhangsan');
select * from t_user;	

验证：read commited

mysql> set global transaction isolation level read committed;

事务A	事务B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;	
	insert into t_user values('zhangsan');
select * from t_user;	
	commit;
select * from t_user;	

验证：repeatable read

mysql> set global transaction isolation level repeatable read;

事务A	事务B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;	
	insert into t_user values('lisi');
	insert into t_user values('wangwu');
	commit;
select * from t_user;	

验证：serializable

mysql> set global transaction isolation level serializable;

事务A	事务B
use bjpowernode;	
	use bjpowernode;
start transaction;	
	start transaction;
select * from t_user;	
insert into t_user values('abc');	
	select * from t_user;

day04

mysql day04课堂笔记

1、索引（index）

1.1、什么是索引?

索引是在数据库表的字段上添加的，是为了提高查询效率存在的一种机制。

一张表的一个字段可以添加一个索引，当然，多个字段联合起来也可以添加索引。

索引相当于一本书的目录，是为了缩小扫描范围而存在的一种机制。

对于一本字典来说，查找某个汉字有两种方式：

第一种方式：一页一页挨着找，直到找到为止，这种查找方式属于全字典扫描。

效率比较低。

第二种方式：先通过目录（索引）去定位一个大概的位置，然后直接定位到这个

位置，做局域性扫描，缩小扫描的范围，快速的查找。这种查找方式属于通过

索引检索，效率较高。

```
t_user

id(idIndex) name(nameIndex) email(emailIndex)
address  (emailAddressIndex)
-----------------------------------------------------
------------------------------

1               zhangsan...
2               lisi
```

```
3                wangwu
4                zhaoliu
5                hanmeimei
6                jack

select * from t_user where name = 'jack';

以上的这条SQL语句会去name字段上扫描，为什么？
    因为查询条件是：name='jack'
```

如果name字段上没有添加索引（目录），或者说没有给name字段创建索引，
MySQL会进行全扫描，会将name字段上的每一个值都比对一遍。效率比较低。

MySQL在查询方面主要就是两种方式：
　第一种方式：全表扫描
　第二种方式：根据索引检索。

注意：
　在实际中，汉语字典前面的目录是排序的，按照a b c d e f....排序，
　为什么排序呢？因为只有排序了才会有区间查找这一说！（缩小扫描范围
　其实就是扫描某个区间罢了！）

　在mysql数据库当中索引也是需要排序的，并且这个索引的排序和TreeSet
　数据结构相同。TreeSet（TreeMap）底层是一个自平衡的二叉树！在mysql
　当中索引是一个B-Tree数据结构。

　遵循左小右大原则存放。采用中序遍历方式遍历取数据。

1.2、索引的实现原理?

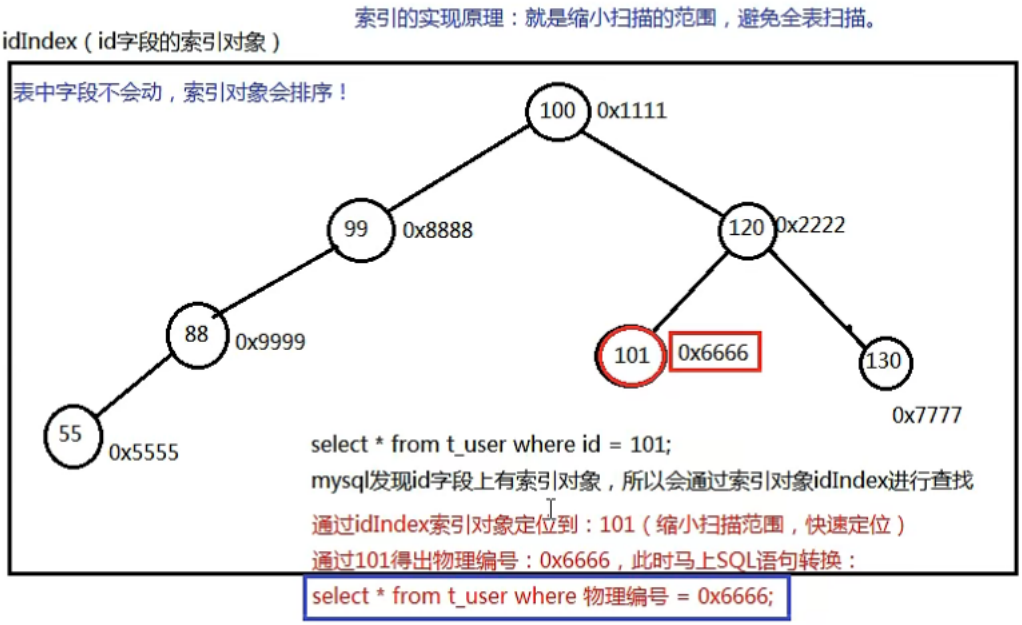
假设有一张用户表：t_user

id(PK)	name	每一行记录在硬盘上都有物理存储编号
100	zhangsan	0x1111
120	lisi	0x2222
99	wangwu	0x8888
88	zhaoliu	0x9999
101	jack	0x6666
55	lucy	0x5555
130	tom	0x7777

提醒1：在任何数据库当中主键上都会自动添加索引对象，id字段上自动有索引，
因为id是PK。另外在mysql当中，一个字段上如果有unique约束的话，也会自动
创建索引对象。

提醒2：在任何数据库当中，任何一张表的任何一条记录在硬盘存储上都有
一个硬盘的物理存储编号。

提醒3：在mysql当中，索引是一个单独的对象，不同的存储引擎以不同的形式
存在，在MyISAM存储引擎中，索引存储在一个.MYI文件中。在InnoDB存储引擎中
索引存储在一个逻辑名称叫做tablespace的当中。在MEMORY存储引擎当中索引
被存储在内存当中。不管索引存储在哪里，索引在mysql当中都是一个树的形式
存在。（自平衡二叉树：B-Tree）



1.3、在mysql当中，主键上，以及unique字段上都会自动添加索引的！！！！

什么条件下，我们会考虑给字段添加索引呢？

条件1：数据量庞大（到底有多么庞大算庞大，这个需要测试，因为每一个硬件环境不同）

条件2：该字段经常出现在where的后面，以条件的形式存在，也就是说这个字段总是被扫描。

条件3：该字段很少的DML(insert delete update)操作。（因为DML之后，索引需要重新排序。）

建议不要随意添加索引，因为索引也是需要维护的，太多的话反而会降低系统的性能。

建议通过主键查询，建议通过unique约束的字段进行查询，效率是比较高的。

1.4、索引怎么创建？怎么删除？语法是什么？

创建索引：

```
    mysql> create index emp_ename_index on
emp(ename);
```

    给emp表的ename字段添加索引，起名：emp_ename_index

删除索引：

```
    mysql> drop index emp_ename_index on emp;
```

    将emp表上的emp_ename_index索引对象删除。

1.5、在mysql当中，怎么查看一个SQL语句是否使用了索引进行检索?

```
mysql> explain select * from emp where ename =
'KING';
+----+-------------+-------+------+---------------+-
-----+---------+------+------+-------------+
| id | select_type | table | type | possible_keys |
key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+-
-----+---------+------+------+-------------+
|  1 | SIMPLE      | emp   | ALL  | NULL          |
NULL | NULL    | NULL |   14 | Using where |
+----+-------------+-------+------+---------------+-
-----+---------+------+------+-------------+
```

扫描14条记录：说明没有使用索引。type=ALL

```
mysql> create index emp_ename_index on emp(ename);

mysql> explain select * from emp where ename =
'KING';
+----+-------------+-------+------+----------------
+-----------------+---------+-------+------+--------
-----+
| id | select_type | table | type | possible_keys
| key             | key_len | ref   | rows | Extra
    |
```

```
+----+-------------+-------+------+----------------
+-----------------+---------+-------+------+--------
-----+
|  1 | SIMPLE      | emp   | ref  | emp_ename_index
| emp_ename_index | 33      | const |    1 | Using
where |
+----+-------------+-------+------+----------------
+-----------------+---------+-------+------+--------
-----+
```

1.6、索引有失效的时候，什么时候索引失效呢?

失效的第1种情况：

select * from emp where ename like '%T';

ename上即使添加了索引，也不会走索引，为什么？

原因是因为模糊匹配当中以“%”开头了！

尽量避免模糊查询的时候以“%”开始。

这是一种优化的手段/策略。

```
mysql> explain select * from emp where ename
like '%T';
+----+-------------+-------+------+-------------
--+------+---------+------+------+-------------+
| id | select_type | table | type |
possible_keys | key  | key_len | ref  | rows | Extra
      |
+----+-------------+-------+------+-------------
--+------+---------+------+------+-------------+
|  1 | SIMPLE      | emp   | ALL  | NULL
| NULL | NULL    | NULL |   14 | Using where |
+----+-------------+-------+------+-------------
--+------+---------+------+------+-------------+
```

失效的第2种情况：

使用or的时候会失效，如果使用or那么要求or两边的条件字段都要有

索引，才会走索引，如果其中一边有一个字段没有索引，那么另一个

字段上的索引也会失效。所以这就是为什么不建议使用or的原因。

```
mysql> explain select * from emp where ename = 'KING' or job = 'MANAGER';
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | emp   | ALL  | emp_ename_index | NULL | NULL    | NULL |   14 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
```

失效的第3种情况：

使用复合索引的时候，没有使用左侧的列查找，索引失效

什么是复合索引？

两个字段，或者更多的字段联合起来添加一个索引，叫做复合索引。

```
create index emp_job_sal_index on emp(job,sal);

mysql> explain select * from emp where job = 'MANAGER';
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
```

```
    | id | select_type | table | type |
possible_keys     | key               | key_len |
ref   | rows | Extra       |
    +----+-------------+-------+------+------------
------+------------------+---------+-------+------
+-------------+
    |  1 | SIMPLE      | emp   | ref  |
emp_job_sal_index | emp_job_sal_index | 30      |
const |    3 | Using where |
    +----+-------------+-------+------+------------
------+------------------+---------+-------+------
+-------------+

    mysql> explain select * from emp where sal =
800;
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
    | id | select_type | table | type |
possible_keys | key  | key_len | ref  | rows | Extra
      |
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL
| NULL | NULL    | NULL |   14 | Using where |
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
```

失效的第4种情况：

在where当中索引列参加了运算，索引失效。

```
    mysql> create index emp_sal_index on emp(sal);

    explain select * from emp where sal = 800;
    +----+-------------+-------+------+------------
--+---------------+---------+-------+------+--------
-----+
```

```
    | id | select_type | table | type |
possible_keys | key           | key_len | ref   |
rows | Extra       |
    +----+-------------+-------+------+------------
--+---------------+---------+-------+------+--------
-----+
    |  1 | SIMPLE      | emp   | ref  |
emp_sal_index | emp_sal_index | 9       | const |
1 | Using where |
    +----+-------------+-------+------+------------
--+---------------+---------+-------+------+--------
-----+

    mysql> explain select * from emp where sal+1 =
800;
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
    | id | select_type | table | type |
possible_keys | key  | key_len | ref  | rows | Extra
      |
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL
| NULL | NULL    | NULL |   14 | Using where |
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
```

失效的第5种情况：

在where当中索引列使用了函数

```
    explain select * from emp where lower(ename) =
'smith';
    +----+-------------+-------+------+------------
--+------+---------+------+------+-------------+
    | id | select_type | table | type |
possible_keys | key  | key_len | ref  | rows | Extra
      |
```

```
    +----+-------------+-------+------+-------------
--+------+---------+------+------+-------------+
    |  1 | SIMPLE      | emp   | ALL  | NULL
| NULL | NULL    | NULL |   14 | Using where |
    +----+-------------+-------+------+-------------
--+------+---------+------+------+-------------+
```

```
失效的第6...
失效的第7...
```

1.7、索引是各种数据库进行优化的重要手段。优化的时候优先考虑的因素就是索引。

索引在数据库当中分了很多类?

单一索引：一个字段上添加索引。

复合索引：两个字段或者更多的字段上添加索引。

主键索引：主键上添加索引。

唯一性索引：具有unique约束的字段上添加索引。

.....

注意：唯一性比较弱的字段上添加索引用处不大。

2、视图(view)

2.1、什么是视图?

v iew:站在不同的角度去看待同一份数据。

2.2、怎么创建视图对象？怎么删除视图对象?

```
表复制:
mysql> create table dept2 as select * from dept;
```

dept2表中的数据：

```
mysql> select * from dept2;
+--------+------------+----------+
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
+--------+------------+----------+
```

创建视图对象：

```
    create view dept2_view as select * from dept2;
```

删除视图对象：

```
    drop view dept2_view;
```

注意：只有DQL语句才能以view的形式创建。

```
    create view view_name as 这里的语句必须是DQL语句;
```

2.3、用视图做什么?

我们可以面向视图对象进行增删改查，对视图对象的增删改查，会导致原表被操作！（视图的特点：通过对视图的操作，会影响到原表数据。）

```
//面向视图查询
select * from dept2_view;

// 面向视图插入
insert into dept2_view(deptno,dname,loc)
values(60,'SALES', 'BEIJING');

// 查询原表数据
mysql> select * from dept2;
+--------+------------+----------+
```

```
| DEPTNO | DNAME      | LOC      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
|     40 | OPERATIONS | BOSTON   |
|     60 | SALES      | BEIJING  |
+--------+------------+----------+

// 面向视图删除
mysql> delete from dept2_view;

// 查询原表数据
mysql> select * from dept2;
Empty set (0.00 sec)
```

```
// 创建视图对象
create view
    emp_dept_view
as
    select
        e.ename,e.sal,d.dname
    from
        emp e
    join
        dept d
    on
        e.deptno = d.deptno;

// 查询视图对象
mysql> select * from emp_dept_view;
+--------+---------+------------+
| ename  | sal     | dname      |
+--------+---------+------------+
| CLARK  | 2450.00 | ACCOUNTING |
| KING   | 5000.00 | ACCOUNTING |
| MILLER | 1300.00 | ACCOUNTING |
```

```
| SMITH  |  800.00 | RESEARCH   |
| JONES  | 2975.00 | RESEARCH   |
| SCOTT  | 3000.00 | RESEARCH   |
| ADAMS  | 1100.00 | RESEARCH   |
| FORD   | 3000.00 | RESEARCH   |
| ALLEN  | 1600.00 | SALES      |
| WARD   | 1250.00 | SALES      |
| MARTIN | 1250.00 | SALES      |
| BLAKE  | 2850.00 | SALES      |
| TURNER | 1500.00 | SALES      |
| JAMES  |  950.00 | SALES      |
+--------+---------+------------+

// 面向视图更新
update emp_dept_view set sal = 1000 where dname =
'ACCOUNTING';

// 原表数据被更新
mysql> select * from emp;
+-------+--------+-----------+------+------------+--
-------+---------+--------+
| EMPNO | ENAME  | JOB       | MGR  | HIREDATE   |
SAL     | COMM    | DEPTNO |
+-------+--------+-----------+------+------------+--
-------+---------+--------+
|  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 |
800.00 |    NULL |     20 |
|  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 |
1600.00 |  300.00 |     30 |
|  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 |
1250.00 |  500.00 |     30 |
|  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 |
2975.00 |    NULL |     20 |
|  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 |
1250.00 | 1400.00 |     30 |
|  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 |
2850.00 |    NULL |     30 |
```

```
|  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 |
1000.00 |    NULL |     10 |
|  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 |
3000.00 |    NULL |     20 |
|  7839 | KING   | PRESIDENT | NULL | 1981-11-17 |
1000.00 |    NULL |     10 |
|  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 |
1500.00 |    0.00 |     30 |
|  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 |
1100.00 |    NULL |     20 |
|  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 |
950.00 |    NULL |     30 |
|  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 |
3000.00 |    NULL |     20 |
|  7934 | MILLER | CLERK     | 7782 | 1982-01-23 |
1000.00 |    NULL |     10 |
+-------+--------+-----------+------+------------+--
-------+---------+--------+
```

2.4、视图对象在实际开发中到底有什么用？《方便，简化开发，利于维护》

```
    create view
        emp_dept_view
    as
        select
            e.ename,e.sal,d.dname
        from
            emp e
        join
            dept d
        on
            e.deptno = d.deptno;
```

假设有一条非常复杂的SQL语句，而这条SQL语句需要在不同的位置上反复使用。
每一次使用这个sql语句的时候都需要重新编写，很长，很麻烦，怎么办？可以把这条复杂的SQL语句以视图对象的形式新建。在需要编写这条SQL语句的位置直接使用视图对象，可以大大简化开发。并且利于后期的维护，因为修改的时候也只需要修改一个位置就行，只需要修改视图对象所映射的SQL语句。
我们以后面向视图开发的时候，使用视图的时候可以像使用table一样。可以对视图进行增删改查等操作。视图不是在内存当中，视图对象也是存储在硬盘上的，不会消失。

再提醒一下：

视图对应的语句**只能是DQL语句。**

但是视图对象创建完成之后，可以对视图进行增删改查等操作。

小插曲：

增删改查，又叫做：CRUD。

CRUD是在公司中程序员之间沟通的术语。一般我们很少说增删改查。

一般都说CRUD。

C:Create（增）
R:Retrive（查：检索）
U:Update（改）
D:Delete（删）

3、DBA常用命令?

重点掌握：

数据的导入和导出（数据的备份）

其它命令了解一下即可。（这个培训日志文档留着，以后忘了，可以打开文档复制粘贴。）

数据导出?

注意：在windows的dos命令窗口中：

mysqldump bjpowernode>D:\bjpowernode.sql -uroot -p123456

```
可以导出指定的表吗？
    mysqldump bjpowernode emp>D:\bjpowernode.sql -uroot -p123456
```

数据导入?

注意：需要先登录到mysql数据库服务器上。

然后创建数据库：create database bjpowernode;

使用数据库：use bjpowernode

然后初始化数据库：source D:\bjpowernode.sql

★★★4、数据库设计三范式

4.1、什么是数据库设计范式?

数据库表的设计依据。教你怎么进行数据库表的设计。

★★★4.2、数据库设计范式共有?

3个。

第一范式：要求任何一张表必须有主键，每一个字段原子性不可再分。

第二范式：建立在第一范式的基础之上，要求所有非主键字段完全依赖主键，
不要产生部分依赖。

第三范式：建立在第二范式的基础之上，要求所有非主键字段直接依赖主键，
不要产生传递依赖。

声明：三范式是面试官经常问的，所以一定要熟记在心！

设计数据库表的时候，按照以上的范式进行，可以避免表中数据的冗余，空间的浪费。

4.3、第一范式

最核心，最重要的范式，所有表的设计都需要满足。
必须有主键，并且每一个字段都是原子性不可再分。

```
学生编号  学生姓名  联系方式
-----------------------------------------
1001          张三        zs@gmail.com,1359999999
1002          李四        ls@gmail.com,13699999999
1001          王五        ww@163.net,13488888888

以上是学生表，满足第一范式吗？
     不满足，第一：没有主键。第二：联系方式可以分为邮箱地址和电话

学生编号(pk)  学生姓名     邮箱地址                联系电话
-----------------------------------------------------
1001                  张三        zs@gmail.com
1359999999
1002                  李四        ls@gmail.com
13699999999
1003                  王五        ww@163.net
13488888888
```

4.4、第二范式：

建立在第一范式的基础之上，
要求所有非主键字段必须完全依赖主键，不要产生部分依赖。

```
学生编号  学生姓名  教师编号  教师姓名
-----------------------------------------------------
```

1001　　张三　　001　　王老师
1002　　李四　　002　　赵老师
1003　　王五　　001　　王老师
1001　　张三　　002　　赵老师

这张表描述了学生和老师的关系：（1个学生可能有多个老师，1个老师有多个学生）
这是非常典型的：多对多关系！

分析以上的表是否满足第一范式？
　　不满足第一范式。

怎么满足第一范式呢？修改

学生编号+教师编号(pk)　　学生姓名　教师姓名

1001　　001　　张三　　王老师
1002　　002　　李四　　赵老师
1003　　001　　王五　　王老师
1001　　002　　张三　　赵老师

学生编号 教师编号，两个字段联合做主键，复合主键（PK：学生编号+教师编号）
经过修改之后，以上的表满足了第一范式。但是满足第二范式吗？
　　不满足，“张三”依赖1001，“王老师”依赖001，显然产生了部分依赖。
　　产生部分依赖有什么缺点？
　　数据冗余了。空间浪费了。“张三”重复了，“王老师”重复了。

为了让以上的表满足第二范式，你需要这样设计：
　　使用三张表来表示多对多的关系！！！！
　　学生表
　　学生编号(pk)　　学生名字

　　1001　　张三
　　1002　　李四

```
    1003                        王五

    教师表
    教师编号(pk)          教师姓名
    ----------------------------------
    001                     王老师
    002                     赵老师

    学生教师关系表
    id(pk)          学生编号(fk)              教师编号
(fk)
    ------------------------------------------------
------
    1                       1001
001
    2                       1002
002
    3                       1003
001
    4                       1001
002
```

```
背口诀：
    多对多怎么设计？
        多对多，三张表，关系表两个外
键！！！！！！！！！！！！！！
```

4.5、第三范式

第三范式建立在第二范式的基础之上

要求所有非主键字段必须直接依赖主键，不要产生传递依赖。

```
学生编号（PK） 学生姓名 班级编号   班级名称
------------------------------------------------
-----
1001            张三        01          一年一班
```

1002	李四	02	一年二班
1003	王五	03	一年三班
1004	赵六	03	一年三班

以上表的设计是描述：班级和学生的关系。很显然是1对多关系！
一个教室中有多个学生。

分析以上表是否满足第一范式？

满足第一范式，有主键。

分析以上表是否满足第二范式？

满足第二范式，因为主键不是复合主键，没有产生部分依赖。主键是单一主键。

分析以上表是否满足第三范式？

第三范式要求：不要产生传递依赖！

一年一班依赖01，01依赖1001，产生了传递依赖。

不符合第三范式的要求。产生了数据的冗余。

那么应该怎么设计一对多呢？

班级表：一

班级编号(pk)	班级名称
01	一年一班
02	一年二班
03	一年三班

学生表：多

学生编号（PK）	学生姓名	班级编号(fk)
1001	张三	01
1002	李四	02
1003	王五	03
1004	赵六	03

背口诀：

一对多，两张表，多的表加外键！！！！！！！！！！！！

4.6、总结表的设计?

一对多：

一对多，两张表，多的表加外键！！！！！！！！！！！！

多对多：

多对多，三张表，关系表两个外键！！！！！！！！！！！！！！

一对一：

一对一放到一张表中不就行了吗？为啥还要拆分表？

在实际的开发中，可能存在一张表字段太多，太庞大。这个时候要拆分表。

一对一怎么设计？

没有拆分表之前：一张表

```
t_user
id  login_name  login_pwd   real_name   email
address........
-------------------------------------------------
-------
1   zhangsan    123         张三      zhangsan@xxx
2   lisi        123         李四      lisi@xxx
            ...
```

这种庞大的表建议拆分为两张：

```
t_login 登录信息表
id(pk)      login_name      login_pwd
---------------------------------
1               zhangsan        123
2               lisi            123
```

t_user 用户详细信息表

```
id(pk)      real_name   email       address.......
login_id(fk+unique) ------------------------------
------------------------------
100      张三            zhangsan@xxx
1
200          李四                lisi@xxx
2
```

口诀：一对一，外键唯一！！！！！！！！！！

4.7、嘱咐一句话：

数据库设计三范式是理论上的。

实践和理论有的时候有偏差。

最终的目的都是为了满足客户的需求，有的时候会拿冗余换执行速度。

因为在sql当中，表和表之间连接次数越多，效率越低。（笛卡尔积）

有的时候可能会存在冗余，但是为了减少表的连接次数，这样做也是合理的，
并且对于开发人员来说，sql语句的编写难度也会降低。

面试的时候把这句话说上：他就不会认为你是初级程序员了！

作业

1

自己做的：

```
select e.ename,e.sal,e.deptno from emp e join
(select max(sal) as maxsal from emp group by deptno)
m on e.sal = m.maxsal;
```

2

```
select e.ename,e.sal,a.* from emp e join (select
deptno,avg(sal) avgsal from emp group by deptno) a
on a.deptno=e.deptno and e.sal>a.avgsal;
```

3

```
select
    e.deptno,avg(s.grade)
from
    emp e
join
    salgrade s
on
    e.sal between s.losal and s.hisal
group by
    e.deptno;
```

4

```
select
    ename,sal
from
    emp
order by
    sal desc
limit 1;
```

```
select ename,sal from emp where sal not in(select
distinct a.sal from emp a join emp b on
a.sal<b.sal);
```

5

```
select deptno, avg(sal) as avgsal from emp group by
deptno having avgsal=(select max(a.avgsal) from
(select deptno,avg(sal) avgsal from emp group by
deptno) as a);
```

```
select deptno,avg(sal) avgsal from emp group by
deptno order by avgsal desc limit 1;
```

6

```
select d.dname from dept d join (select
deptno,avg(sal) avgsal from emp group by deptno
order by avgsal desc limit 1) e on d.deptno =
e.deptno;
```

7

```
select d.dname from dept d join (select
deptno,avg(sal) avgsal from emp group by deptno
order by avgsal limit 1) e on d.deptno = e.deptno;//
平均薪资最低的就是等级最低的
```

```
select t.deptno,t.avgsal,s.grade from (select
e.deptno,avg(e.sal) avgsal from emp e group by
deptno) t join salgrade s on t.avgsal between
s.losal and s.hisal; //先求出等级

select d.dname,a.grade from (select
t.deptno,t.avgsal,s.grade from (select
e.deptno,avg(e.sal) avgsal from emp e group by
deptno) t join salgrade s on t.avgsal between
s.losal and s.hisal) as a join dept as d on a.deptno
= d.deptno order by grade limit 1; //再求最终结果
```

8

```
//先求普通员工-除了领导剩下的都是普通员工
//领导
select distinct mgr from emp;
//员工最高薪资
//not in在使用的时候，后面小括号中记得排除NULL。
select max(sal) as maxsal from emp where empno not
in (select distinct mgr from emp where mgr is not
null);
//最终结果
select ename,sal from emp where sal>(select max(sal)
as maxsal from emp where empno not in (select
distinct mgr from emp where mgr is not null));
```

9

```
select ename,sal from emp order by sal desc limit 5;
```

10

```
select ename,sal from emp order by sal desc limit 5,
5;
```

11

```
select ename,hiredate from emp order by hiredate
desc limit 5;
```

12

```
select count(*),t.grade from (select s.grade,e.ename
from emp e join salgrade s on e.sal between s.losal
and s.hisal ) t group by t.grade;
```

13

```
//先找出黎明老师的课号
select cno from c where cteacher = '黎明';
//找出选了黎明老师课的学生编号
select sno from sc where cno = (select cno from c
where cteacher = '黎明');
//再找出除此之外的学生编号
select sname from s where sno not in (select sno
from sc where cno = (select cno from c where
cteacher = '黎明'));
```

```
//先找出不及格成绩及对应学生编号
select sno,scgrade from sc where scgrade<60;
//再按学生编号分组，统计每个学生不及格门数、平均成绩
select e.sno,count(*) as num,sum(e.scgrade)/count(*)
as avggrade from (select sno,scgrade from sc where
scgrade<60) as e group by sno;
//找到门数大于等于2的人的学生编号
select p.sno,p.avggrade from (select e.sno,count(*)
as num,sum(e.scgrade)/count(*) as avggrade from
(select sno,scgrade from sc where scgrade<60) as e
group by sno) as p where p.num>=2;
//最终找到姓名及平均成绩
select s.sname,f.avggrade from s as s join (select
p.sno,p.avggrade from (select e.sno,count(*) as
num,sum(e.scgrade)/count(*) as avggrade from (select
sno,scgrade from sc where scgrade<60) as e group by
sno) as p where p.num>=2) f on s.sno = f.sno;
```

```
//先找出学过1号课的学生编号
select sno from sc where cno = 1;
//再找出学过2号课的学生编号
select sno from sc where cno =2;
//都学过的学生编号
select a.sno from (select sno from sc where cno = 1)
a join (select sno from sc where cno =2) b on a.sno
= b.sno;
//最终结果
select s.sname from s as s join (select a.sno from
(select sno from sc where cno = 1) a join (select
sno from sc where cno =2) b on a.sno = b.sno) c
where s.sno = c.sno;
```

14

```
//左外连接
select e1.ename '员工',e2.ename '领导' from emp e1
left join emp e2 on e1.mgr=e2.empno;
```

15

```
select a.ename '员工',a.hiredate,b.ename '领
导',b.hiredate,d.dname from emp a join emp b on
a.mgr=b.empno join dept d on a.deptno = d.deptno
where a.hiredate < b.hiredate;
```

16

```
select d.dname,e.* from emp e right join dept d on
e.deptno=d.deptno;
```

17

```
select a.dname,a.num from (select t.dname,count(*)
as num from (select d.dname,e.* from emp e right
join dept d on e.deptno=d.deptno) t group by dname)
a where a.num>=5;
```

18

```
select e.* from emp e where e.sal>(select sal as
smithsal from emp where ename ='smith');
```

19

```
//各部门的人数
select d.dname,count(*) as num from emp e join dept
d on d.deptno=e.deptno group by e.deptno;
//所有办事员的姓名
select e.ename,d.dname from emp e join dept d on
d.deptno=e.deptno where e.job ='CLERK';
//最终
select a.ename,a.dname,b.num from (select
e.ename,d.dname from emp e join dept d on
d.deptno=e.deptno where e.job ='CLERK') as a join
(select d.dname,count(*) as num from emp e join dept
d on d.deptno=e.deptno group by e.deptno) as b on
a.dname = b.dname;
```

20

```
// 列出各种工作的最低薪金
select min(sal) as minsal, job from emp group by
job;
//列出各种工作人数
select job,count(*) as num from emp group by job;
//最终结果
select a.job,b.num from (select min(sal) as minsal,
job from emp group by job) a join (select
job,count(*) as num from emp group by job) b on
a.job = b.job where a.minsal>1500;
```

21

```
select ename from emp where deptno = (select deptno
from dept where dname = 'SALES');
```

22

```
select
```

```
    e.ename '员工',d.dname,a.ename '领
导',e.sal,s.grade
from
    emp e
join
    dept d
on
    e.deptno = d.deptno
left join
    emp a
on
    e.mgr=a.empno
join
    salgrade s
on
    e.sal between s.losal and s.hisal
where
    e.sal > (select avg(sal) from emp);
```

23

```
select
    e.ename,e.job,d.dname
from
    emp e
join
    dept d
on
    e.deptno = d.deptno
where
    e.job = (select job from emp where ename =
'SCOTT')
and
    e.ename <> 'SCOTT';
```

24

```
//注意用in
select
    e.ename,e.sal
from
    emp e
where
    e.sal in (select sal from emp where deptno=30)
and
    e.deptno != 30;
```

25

```
select
    e.ename,e.sal,d.dname
from
    emp e
join
    dept d
on
    e.deptno=d.deptno
where
    e.sal > (select max(sal) from emp where deptno =
30);
```

26

```
select
    d.deptno,d.dname,count(e.ename) as '员工数
量',ifnull(avg(e.sal),0) as
avgsal,ifnull(avg(timestampdiff(YEAR,e.hiredate,now(
))),0) as avgservicetime
from
    emp e
right join
    dept d
on
    e.deptno= d.deptno
group by
    d.deptno;
```

在mysql当中怎么计算两个日期的“年差”，差了多少年?
TimeStampDiff(间隔类型, 前一个日期, 后一个日期)

timestampdiff(YEAR, hiredate, now())

间隔类型:
SECOND 秒,
MINUTE 分钟,
HOUR 小时,
DAY 天,
WEEK 星期
MONTH 月,
QUARTER 季度,
YEAR 年

27

```
select
    e.ename,d.dname,e.sal
from
    emp e
join
    dept d
on
    e.deptno=d.deptno;
```

28

```
select
    d.*,count(e.ename) as '人数'
from
    emp e
right join
    dept d
on
    d.deptno=e.deptno
group by
    d.deptno;
```

29

```sql
select
    e.ename,t.*
from
    emp e
join
    (select
        job,min(sal) as minsal
    from
        emp
    group by
        job) t
on
    e.job = t.job and e.sal = t.minsal;
```

30

```sql
select
    deptno,min(sal)
from
    emp
where
    job = 'manager'
group by
    deptno;
```

31

```sql
select
    ename,(sal+ifnull(comm,0))*12 as yearsal
from
    emp
order by
    yearsal asc;
```

32

```
select
    a.ename '员工',b.ename '领导'
from
    emp a
join
    emp b
on
    a.mgr=b.empno
where
    b.sal>3000;
```

33

```
select

d.deptno,d.dname,d.loc,count(e.ename),ifnull(sum(e.sal),0) as sumsal
from
    emp e
right join
    dept d
on
    e.deptno = d.deptno
where
    d.dname like '%S%'
group by
    d.deptno,d.dname,d.loc;
```

34

```
update emp set sal = sal * 1.1 where timestampdiff(YEAR, hiredate, now()) > 30;
```